Chapter 5

D_S^+ and D^+ signal extraction

The main ingredient for the evaluation of the D_s^+/D^+ production-yield ratio is the D_s^+ and D^+ raw yield, i.e., the number of reconstructed D_s^+ and D^+ mesons. Due to the vast amount of combinatorial background and the limited efficiency of about 1%, the extraction of the raw yield is not possible through a candidate counting method. Instead, the raw yield is obtained on a statistical basis by fitting the invariant-mass distribution of the D_s^+ and D^+ candidates passing tight selection criteria. To reduce the combinatorial background and enhance the efficiency of D-meson selection, Machine Learning algorithms have been employed. The following sections describe the procedure for the extraction of the raw yield of D_s^+ and D^+ mesons.

5.1 Machine Learning

The term *Machine Learning* (ML) is a broad and versatile concept, encompassing a wide range of algorithms that grant computers the capacity to learn and adapt without being explicitly programmed to do so [1]. A more comprehensive definition characterises ML as the study of algorithms that enhance their performance (P) at a specific task (T) through the accumulation of experience (E) [2]. In recent years, ML techniques have witnessed widespread adoption across diverse fields, with significant impacts realised especially with the emergence of generative models such as GPT [3]. ML algorithms have found extensive applications in the high-energy physics field, primarily for the task of distinguishing interesting signals from the vast background present in particle collision data. Furthermore, these algorithms have been employed as triggers, aiding in the rapid identification of events of interest, and have also been instrumental in event reconstruction. Notably, ML algorithms were used in the discovery of the Higgs boson [4], one of the most significant achievements in the field of particle physics. Aggiungere che permette di fare selezioni non lineari

5.1.1 Supervised learning

Supervised learning is one of the main branches of machine-learning problems, together with unsupervised and reinforcement learning. Machine learning tasks are usually described in terms of how the machine learning system should process an

example, which is a collection of features $\mathbf{x}$ that have been quantitatively measured from some object or event that one wants the machine learning system to process. In the case of supervised learning, each example is coupled with a corresponding label or target, $\mathbf{y}$. The objective of supervised learning is to learn to predict or infer $\mathbf{y}$ based on the associated features, $\mathbf{x}$, assuming that there exists a functional relationship $\mathbf{y} = f(\mathbf{x})$ between the two. The goal of the machine learning system is to produce an approximation $\hat{f}(\mathbf{x})$ of the true function $f(\mathbf{x})$ by minimising a given loss function, which quantifies the discrepancy between the predicted and true labels. Supervised learning problems exhibit further segmentation into two distinct sub-categories, known as classification and regression. In the former, the label $\mathbf{y}$ assumes values from a finite and discrete set of categories, often representing distinct classes or groups. In the latter, the label $\mathbf{y}$ takes the form of one or more continuous variables. This necessitates the learning system to deduce a continuous function or mapping between $\mathbf{x}$ and $\mathbf{y}$, where the goal is to predict and approximate numerical values or class affiliations.

The application of a supervised learning algorithm to a dataset involves the following steps: i. the model is trained on a set of labelled data, i.e., the value of $\mathbf{y}$ is known for each example in the training set; ii. the model is tested on a separate set of labelled data, known as the test set, to evaluate its performance; iii. the model is then used to make predictions on new, unseen data.

Training

During the training process, the model learns (i.e., adjusts its internal parameters) to map the input features $\mathbf{x}$ to the corresponding labels $\mathbf{y}$ by minimizing a given loss function. Typically used loss functions include the Mean Squared Error (MSE) for regression tasks and the Cross-Entropy loss for classification tasks. The loss function is minimised through an optimization algorithm, usually stochastic gradient descent [5], which iteratively updates the model parameters to reduce the loss. Since an overoptimisation of the model on the training data can lead to poor generalization on unseen data (the model is said to be *overfitting*), a regularisation term is often added to the loss function to penalise overly complex models. The training process continues until the model reaches a satisfactory level of performance on the training data, or until its performance does not improve further.

Before the final model training, hyperparameters tuning is performed to optimise the model's performance. *Hyperparameters* are parameters that are not learned during the training process, but rather define the model's architecture and the training process itself. Hyperparameters tuning is usually performed through a grid search or random searchaggiungere bayesian, where different combinations of hyperparameters are tested on a dedicated labelled dataset, different from the training set: the validation set. Models with different hyperparameter sets are trained with a reduced training phase, and those yielding the best performance are then selected for the final model training.

Testing

After the model has been trained, its performance is evaluated on a dataset that was not used during the training process, known as the test set. Like the training and validation sets, also the test set contains labelled examples. While during the training the model is optimised to minimise the loss function, the test set is used to estimate the model's generalization error, i.e., how well the model performs on unseen data. The model's performance is evaluated using metrics that are specific to the task at hand, such as accuracy for classification tasks, or Mean Squared Error (MSE) for regression tasks. Once the model achieves satisfactory performance on the test set, it is ready to be used for making predictions on unlabelled data.

Cross-validation

With the strategy defined above to optimise the hyperparameters, train the model and validate its performance, the dataset is divided into three subsets: the training set, the validation set, and the test set. When small datasets are involved, this division can lead to a suboptimal model, as the model's performance can be highly dependent on the specific examples in the training, validation, and test sets. Furthermore, this approach limits the amount of data available for training the model, which can lead to poor generalization. To mitigate this issue, a technique called cross-validation is often employed. It consists on dividing the training sample into k subsets of equal size, called *folds*. Then, the ML algorithm is trained k times, each time using $k-1$ folds as training set, while the remaining fold is used as validation set. The model's performance is then averaged over the k folds to obtain a more robust estimate of this quantity. This operation is repeated for each hyperparameter configuration to be considered. The hyperparameter configuration minimising the loss function is then chosen as the optimal configuration.

5.2 D_s^+ and D^+ selection using Machine Learning

The task of extracting D_s^+ and D^+ signals from the vast combinatorial background is a challenging one, due to the large amount of background compared to signal. It is however an excellent example of classification problem, and ML algorithms can therefore be exploited to enhance the efficiency of the selection.

5.2.1 Data preparation

Aggiungere come settiamo le labels (0=bkg, 1 = prompt, 2= FD) In order to train a ML model, a labelled dataset with a well defined set of features is required. The dataset is composed of signal and background examples. To obtain a pure sample of signal candidates, Monte Carlo simulations are used to generate D_s^+ and D^+ mesons. Proton-proton collisions are generated using the PYTHIA 8 event generator [6] with colour-reconnection Mode 2 [7], and the generated particles are propagated through the ALICE detector using the GEANT4 transport simulation toolkit [8]. To enrich the sample of heavy-flavour hadrons, $c\bar{c}$ and $b\bar{b}$ pairs are injected into each simulated event.

Only prompt and non-prompt D_s^+ mesons are used to train the model, as D^+ mesons decay into the same final state as D_s^+ mesons, and selections optimised to reconstruct D_s^+ mesons are also effective for D^+ mesons.

Background candidates are obtained from real data, as MC simulations may not able to reproduce the complexity of soft processes occurring in the underlying event, or may not be able to model the detector response accurately. The background examples are obtained by selecting candidates from a subsample of the full data sample (correspondig to its 3%) in an invariant-mass region away from both the D_s^+ and D^+ mass peaks, where $1.7 < M < 1.75$ GeV/c^2 or $2.1 < M < 2.15$ GeV/c^2, as shown in Fig. 5.1.

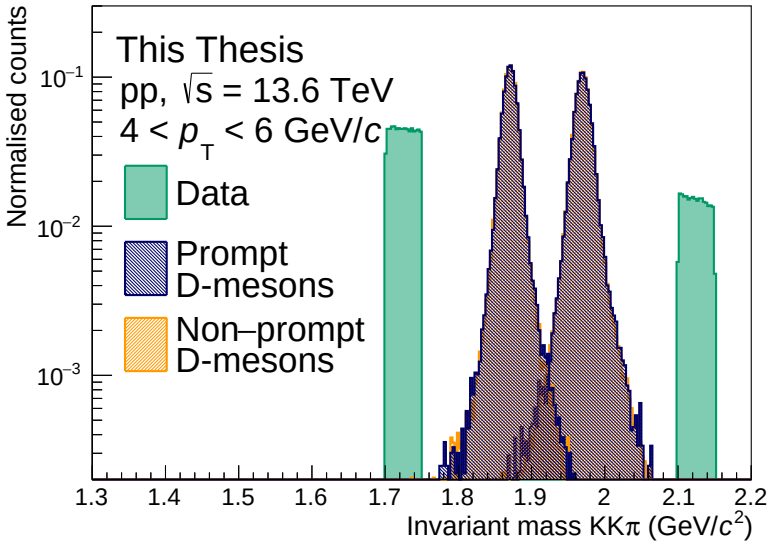

Figure 5.1: Invariant mass distribution of the backgound candidates used to train the ML model (green) and of prompt and non-prompt D-mesons (blue and orange, respectively), taken from Monte Carlo simulations, in the $4 < p_{\mathrm{T}} < 6$ GeV/c interval. Background candidates are selected in the $1.7 < M < 1.75$ GeV/c^2 or $2.1 < M < 2.15$ GeV/c^2 invariant-mass interval.

The dataset is then divided into two different subsamples. The first comprehends 80% of the data, and is used to train the model, while the remaining 20% is used to test its performance. In addition, since the D-mesons decay topology can significantly differ depending on the p_{T} of the meson due to different Lorentz boosts, the dataset is divided into several p_{T} intervals, and the model is trained and tested separately for each of them. The total number of candidates availabe for training and testing the model is reported in Table 5.1 for the considered p_{T} intervals.

To produce a balanced dataset, the number of candidates in each class is equalised to the number of examples in the minority class. This is achieved by randomly selecting a subset of the majority classes. The balanced dataset is then used to train the model.

The choice of features used to separate signal from background is crucial, as they must be able to discriminate between signal and background candidates, and must be chosen in such a way that no bias is introduced in the final result. The variables

Table 5.1: Number of candidates within the p_{T} intervals used to train and test the model.

p_{T} (GeV/c)	Prompt $\mathrm{D_s^+}$	Non-prompt $\mathrm{D_s^+}$	Background
0–1.5	$\sim 4.6 \times 10^3$	$\sim 21 \times 10^3$	$\sim 726 \times 10^3$
1.5–2	$\sim 6.1 \times 10^3$	$\sim 24 \times 10^3$	$\sim 92 \times 10^3$
2–3	$\sim 26 \times 10^3$	$\sim 96 \times 10^3$	$\sim 123 \times 10^3$
3–4	$\sim 34 \times 10^3$	$\sim 124 \times 10^3$	$\sim 114 \times 10^3$
4–5	$\sim 31 \times 10^3$	$\sim 113 \times 10^3$	$\sim 63 \times 10^3$
5–6	$\sim 24 \times 10^3$	$\sim 89 \times 10^3$	$\sim 29 \times 10^3$
6–8	$\sim 32 \times 10^3$	$\sim 115 \times 10^3$	$\sim 22 \times 10^3$
8–12	$\sim 23 \times 10^3$	$\sim 89 \times 10^3$	$\sim 10 \times 10^3$
12–24	$\sim 9.7 \times 10^3$	$\sim 39 \times 10^3$	$\sim 2.6 \times 10^3$

used to train the model were introduced in Chapter **??**, and are a mix of topological, kinematic, and PID variables. The key idea is to exploit the displaced topology of the D-meson decay, which is a distinctive feature of the signal candidates, the kinematic properties of the D-meson decay, and the PID information of the daughter tracks to discriminate between signal and background candidates. The features used to train the model are reported in Table 5.2. The number in parenthesis after $n\sigma$ indicates the prong number.

Table 5.2: Candidate features used to train the ML model.

Variable
$\cos\theta_p$
$\cos\theta_p^{xy}$
Decay length
Decay length XY
Candidate impact parameter XY
$\lvert\cos^3\theta'(\mathrm{K})\rvert$
Prong 0 impact parameter XY
Prong 1 impact parameter XY
Prong 2 impact parameter XY
$n\sigma^{\pi}_{comb}(0)$
$n\sigma^{\pi}_{comb}(1)$
$n\sigma^{\pi}_{comb}(2)$
$n\sigma^{\mathrm{K}}_{comb}(0)$
$n\sigma^{\mathrm{K}}_{comb}(1)$
$n\sigma^{\mathrm{K}}_{comb}(2)$

The invariant mass of the candidate and its p_{T} are not used to train the model. Exploiting such variables would introduce a bias in the final result, as the model

would be trained to select candidates within a specific invariant mass region (that of $\mathrm{D_s^+}$ and $\mathrm{D^+}$ mesons) or p_{T}. This would affect both the selection of the candidates and the p_{T} distribution of the final sample, leading to a biased p_{T}-differential yield. However, some of the variables used to train the model may be correlated with the invariant mass of the candidate, and the ML may learn to discriminate the signal from the background by exploiting the correlation with the $\mathrm{D_s^+}$ meson mass and transverse momentum, rather than the physical properties of the signal and background. To exclude this possibility, the correlation between the features used to train the model is studied. To quantitatively describe the correlation between the variables, the Pearson correlation coefficient ρ is evaluated for each pair of variables. It is defined as the ratio between the covariance of two variables and the product of their standard deviations, $\rho(x, y) = \mathrm{cov}(x, y)/(\sigma_x \sigma_y)$. It expresses the strength and direction of a linear correlation between two variables, ranging from $\rho = 1$ (perfect positive linear correlation) to $\rho = -1$ (perfect negative linear relationship). $\rho = 0$ indicates no linear correlation.

The correlation matrix of the features used to train the model is shown in Fig. 5.2 for the prompt $\mathrm{D_s^+}$, non-prompt $\mathrm{D_s^+}$ and background classes, in the $2 < p_{\mathrm{T}} < 3$ GeV/c. The correlation with the invariant mass and the transverse momentum is also reported. The Pearson coefficient is encoded in the colour of the cell, with red indicating a positive correlation, blue a negative correlation, and grey no correlation. The correlation matrix shows that the variables used to train the model are not correlated with the invariant mass of the candidate, suggesting that a ML should not modify the invariant-mass distribution of the selected candidates, which would introduce a bias in the measurement.

Variables carrying the same physical information, such as those related to the candidate decay length, pointing angle, and impact parameter, are strongly correlated among each other, as expected. Different degrees of correlation between the same variables in the different classes are observed. The ML model can exploit these differences to discriminate between signal and background candidates.

5.2.2 Boosted Decision Trees

Once the training dataset has been composed, and the features have been selected, the ML architecture has to be chosen. Several algorithms are available, each with its own strengths and weaknesses. The choice of the algorithm depends on the specific problem to solve, the size of the dataset, and the computational resources available.

Boosted decision trees [9, 10] (BDTs) are a family of machine learning algorithms employed in different fields, including high-energy physics. Their building blocks are decision trees, which are a versatile type of supervised learning algorithm that can be used for both classification and regression tasks. A decision tree is made of many *nodes*, each containing conditions that split the data into two [11] or more [12] children nodes. The first node of the tree, which receives all the data, is called the *root*, while nodes that do not further split the data are called *leaves*, and contain the output of the tree. The model is trained by considering the Gini index, which

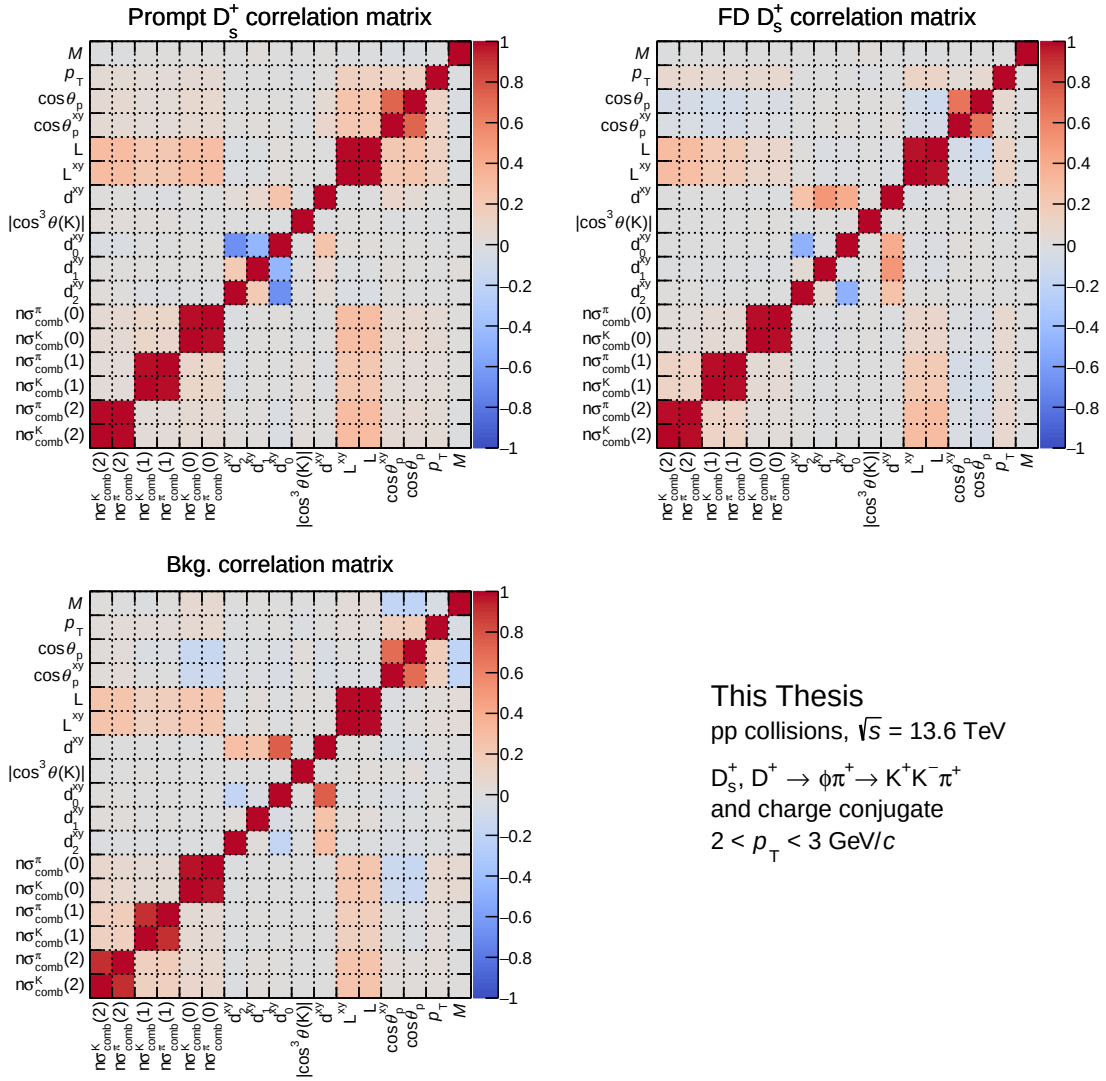

Figure 5.2: Correlation matrix of the features used to train the ML model for prompt D_s^+ (top-left), non-prompt D_s^+ (top-right), and background (bottom-left) candidates in the $2 < p_T < 3$ GeV/c interval. The correlation with the invariant mass and the transverse momentum is also reported. The Pearson coefficient is encoded in the colour of the cell, with red indicating a positive correlation, blue a negative correlation, and grey no linear correlation.

measures the impurity of the node:

$$G = 1 - \sum_{i=1}^{n} p_i^2 \quad ,$$

where p_i is the fraction of samples in the node that belong to class i. The Gini index therefore provides an indication of the quality of the split. A commonly used algorithm to build *binary* decision trees (i.e., each node contains binary-output conditions, and is split into two children nodes) is the *Classification And Regression Tree* (CART) algorithm [11], which recursively splits the dataset into subsets based on a single feature k and a threshold t_k that minimises the impurity of the subsets (weighted by their size). The cost function that the algorithm tries to minimise is given by

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}} \quad ,$$

where m_{left} and m_{right} are the number of samples in the left and right nodes, respectively, summing up to the total number of samples m, and G_{left} and G_{right} are the Gini indices of the left and right nodes. The tree is grown until a stopping criterion is met, such as a maximum depth, a minimum number of samples in a node, or a minimum impurity decrease. These are all hyperparameters that can be tuned to optimise the model's performance.

Given their simplicity, decision trees are fairly easy to interpret, and are often called *white-box* models (in contrast to BDTs and neural networks, where the decision-making process is less transparent, therefore called *black-box* models). An additional strength of decision trees is that they require very little data preparation, e.g., they do not require feature scaling or centering, making them a very powerful yet simple tool for data analysis. However, they are prone to overfitting, as they can grow to a large depth, capturing the noise in the training data. To mitigate this issue, their depth is usually constrained, but this may lead to a model with limited discrimination power. To build a robust model with a good discrimination power, ensemble methods may be used. Several decision trees can be trained, and the final prediction is obtained by combining the outcome of all the trees.

XGBoost

In this work, the Extrame Gradient Boosting [13] (XGBoost) Boosted Decision Trees (BDT) algorithm is used. It has achieved state-of-the-art results in a number of machine learning and data mining challenges (for example in Ref. [14]). In addition, this algorithm, which is available as an open-source package, can be easily parallelised on CPUs and GPUs [15] reducing the training and application time.

The term *boosting* refers to any ensemble method combining several weak learners into a strong learner. The general idea of most boosting methods is to train many predictors sequentially, each trying to correct its predecessor [16]. The function estimate $\hat{f}(\mathbf{x})$ is parametrised with an additive functional form:

$$\hat{f}(x) = \sum_{\mathrm{k}=1}^{\mathrm{M}} \hat{f}_{\mathrm{k}}(x) \quad ,$$

where M is the number of iterations, $\hat{f}_0(x)$ is the initial prediction, and $\hat{f}_{\mathrm{i}}(x)$ is the function increment at the i-th iteration, also called *boost*. To reduce the loss function, a new weak learner, whose functional form is parametrised as $h(\mathbf{x}, \theta)$, can be added to the ensemble:

$$\hat{f}_{\mathrm{t}}(\mathbf{x}) \leftarrow \hat{f}_{\mathrm{t}-1}(\mathbf{x}) + \rho_{\mathrm{t}} h(\mathbf{x}, \theta_{\mathrm{t}}) \quad ,$$

ρ_t is the step size, which is optimised for each iteration t, together with the parameters θ_{t} of the weak learner:

$$(\rho_{\mathrm{t}}, \theta_{\mathrm{t}}) = \arg\min_{\rho,\theta} \sum_{i=1}^{\mathrm{N}} L\left(y_i, \hat{f}_{\mathrm{t}-1}(x_i) + \rho h(x_i, \theta)\right) \quad ,$$

where L is the loss function, and y_i is the true label of the i-th example. Despite having a well-defined set of equations for minimising the loss function, the optimisation of the parameters is not trivial, as the loss function is non-convex and the search space is high-dimensional. Therefore, the optimisation is usually performed using a gradient-based algorithm [9, 17], where $h(\mathbf{x}, \theta_{\mathrm{t}})$ is chosen as the most parallel function to the negative gradient of the loss function with respect to the previous prediction $g_{\mathrm{t}}(\mathbf{x})$:

$$g_{\mathrm{t}}(\mathbf{x}) = E_{\mathbf{y}}\left[\frac{\partial L(\mathbf{y}, \hat{f}_{\mathrm{t}-1}(\mathbf{x}))}{\partial \hat{f}_{\mathrm{t}-1}(\mathbf{x})}\middle|\mathbf{x}\right] \quad ,$$

where $E_{\mathbf{y}}$ is the expectation over the true labels. The parameters are then optimised by minimising the difference between the negative gradient and the weak learner prediction:

$$(\rho_{\mathrm{t}}, \theta_{\mathrm{t}}) = \arg\min_{\rho,\theta} \sum_{\mathrm{i}=1}^{\mathrm{N}} \left[-g_{\mathrm{t}} - \rho h(x_{\mathrm{i}}, \theta)\right]^2 \quad .$$

Through the iterative addition of weak learners, the model is able to learn complex patterns in the data, and to reduce the loss function. The final prediction is obtained by summing the predictions of all the weak learners. In the XGBoost algorithm, the weak learners are decision trees.

5.2.3 Tuning the model's hyperparameters

The XGBoost algorithm has several hyperparameters [18] that can be tuned to optimise the model's performance. The most important hyperparameters are:

- `eta` or `learning_rate`, which is the step size shrinkage of the gradient descent algorithm. To reduce the risk of overfitting, this factor multiplies the weak-learner prediction ($\rho_{\mathrm{t}} h(x_{\mathrm{i}}, \theta) \rightarrow \texttt{eta} \cdot \rho_{\mathrm{t}} h(x_{\mathrm{i}}, \theta)$), and is usually set to a small value, such as 0.3;

- `max_depth`, which is the maximum depth of the tree. A large depth can lead to overfitting, while a small depth can lead to a model with limited discrimination power;

- `n_estimators`, which defines the number of trees to train. A large number of weak learners can lead to overfitting, while a small number can lead to a model with limited discrimination power. Usually, the number of weak learners is set to around 1000;
- `subsample`, which is the fraction of the training data to be used to train each tree at each iteration;
- `min_child_weight`, which is the minimum sum of instance weight needed in a child. It is related to the purity in a node, and it is used to stop the tree growth;
- `colsample_bytree`, which is the fraction of features to be used to train each tree at each iteration;
- `tree_method`, which defines the algorithm used to build the trees. The `hist` option uses an optimised histogram-based algorithm and is usually the fastest.

The hyperparameters are optimised using the Optuna framework [19], which proved to be a powerful tool thanks to its state-of-the-art algorithms for sampling the hyperparameter space and for efficiently pruning unpromising trials. The Tree-Structured Parzen Estimator [20], is used in this Thesis. It is a Bayesian optimisation [21, 22] algorithm able to explore the hyperparameter space efficiently. The aim of a Bayesian optimisation is to maximise (or minimise, depending on the task) an objective function $f(\mathbf{x})$ by iteratively sampling a bounded hyperparameter space, χ. The algorithm builds a probabilistic model of the objective function, and uses it to decide which hyperparameters to sample next. The model is updated at each iteration, and the hyperparameters that are most likely to improve the model's performance are sampled. The Optuna algorithm is also able to prune unpromising trials, reducing the computational cost of the optimisation. The optimisation is performed using a 5-fold cross-validation, and the hyperparameters that maximise the macro-averaged one-vs-one ROC AUC metric (described in detail in Sec. **??**) are chosen as the optimal configuration. The hyperparameters optimised for the XGBoost model are reported in Table 5.3. An additional hyperparameter, `lambda`, which is the L2 regularisation term, is also optimised. It helps to prevent overfitting by penalising overly complex models. The optimal hyperparameters are then used to train the model on the full training dataset.

5.2.4 Evaluation of the model's performance

After training the model, its performance is evaluated on the test dataset. The model's performance can be assessed using a *confusion matrix*, which summarises the number of examples for a given class (the true label) that are classified by the model as belonging to any of the available classes (the predicted label). A good model should provide a high number of correctly-classified examples (reported on the diagonal of the confusion matrix), and a low number of misclassified examples (off-diagonal elements of the confusion matrix). The confusion matrix also allows an understanding of which classes are more difficult to classify, and which classes

Table 5.3: Optimised hyperparameter configuration for the p_T bins considered in the model training.

Hyper-parameter	p_T interval (GeV/c)								
	0–1.5	1.5–2	2–3	3–4	4–5	5–6	6–8	8–12	12–24
max_depth	3	3	3	3	3	3	3	3	3
learning_rate	0.04	0.068	0.065	0.10	0.091	0.84	0.070	0.046	0.030
n_estimators	473	339	1352	909	1256	1392	1142	1437	1188
min_child_weight	1	3	10	10	10	9	3	7	5
subsample	0.87	0.95	0.84	0.85	0.95	0.85	0.81	0.94	0.88
colsample_bytree	0.91	0.98	0.90	0.98	0.96	0.95	0.88	0.96	0.89
lambda	8.0×10^{-4}	4.8×10^{-4}	9.1×10^{-4}	1.4×10^{-4}	3.0×10^{-4}	3.2×10^{-4}	1.9×10^{-4}	9.8×10^{-4}	6.7×10^{-4}
tree_method	hist	hist	hist	hist	hist	hist	hist	hist	hist

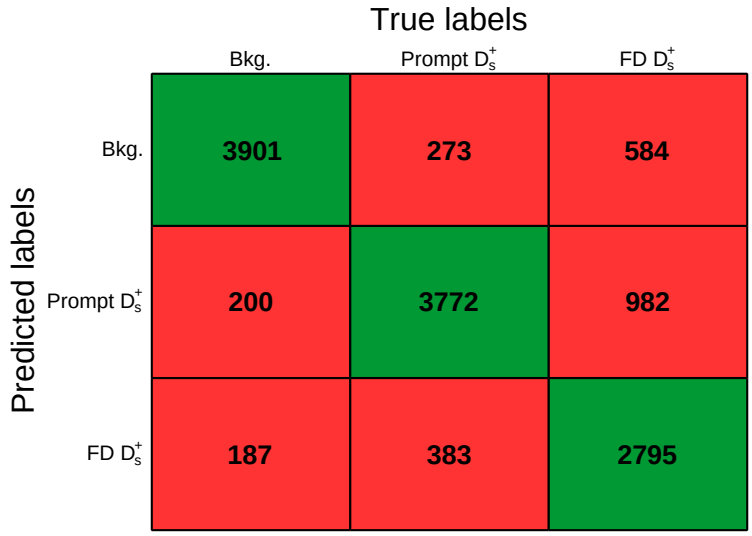

Figure 5.3: Confusion matrix for the XGBoost model trained on the $2 < p_T < 3$ GeV/c interval. The classes are assigned as follows: if a candidate score is less than 0.5, it is classified as background; if it is greater than 1.5, it is classified as non-prompt D_s^+, and it is classified as prompt D_s^+ otherwise.

are more likely to be confused with each other. An example of confusion matrix for the XGBoost model trained on the $2 < p_T < 3$ GeV/c interval is shown in Fig. 5.3.

Despite providing a lot of information on the model's performance, more concise metrics of the model's performance are usually used, for a more direct comparison between different models. In addition, the confusion matrix provides a threshold-dependent measure of the model's performance, as the classification threshold can be varied to increase the number of correctly classified signal candidates at the expense of the number of correctly classified background candidates, and vice versa.

In binary classification tasks, where only two classes are available (a positive and a negative class), several metrics can be defined from the elements of the confusion matrix. The 2×2 confusion matrix contains four entries: the true positives (TP), which are the number of correctly classified positive candidates, the false positives (FP), which are the number of negative candidates being mistakenly classified as positives, and the analogously defined true negatives (TN) and false negatives (FN).

One of the most used tools for binary classifiers is the *Receiver Operating Characteristic* (ROC) curve, which represents the true positive rate (TPR) against the false positive rate (FPR) for different threshold values. The TPR is the fraction of correctly classified positive candidates ($\mathrm{TPR} = \mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$), while the FPR is the fraction of incorrectly classified negative candidates ($\mathrm{FPR} = \mathrm{FP}/(\mathrm{FP} + \mathrm{TN})$). If positive candidates are selected as those with a score greater than a certain threshold t, then when $t = 0$ all candidates are classified as positive, and both the TPR and FPR will be equal to 1. On the other hand, if $t = 1$, no candidate is classified as positive, and the TPR and FPR will both equal 0. The ROC *Area Under the Curve* (AUC), is used to measure the model's ability to discriminate between positive and negative candidates, for any given threshold. The ROC AUC ranges from 0 to 1. A random classifier has a ROC AUC of 0.5, while a perfect classifier has a ROC AUC of 1. The ROC AUC is a threshold-independent measure of the model's performance, and is used to compare different models.

In a multiclass classification task, where more than two classes are available, the ROC curve and the ROC AUC cannot be used. In this case, the ROC curve can be generalised to the *One-vs-One* ROC curve, which is a plot of the TPR against the FPR for each class. The ROC AUC can be averaged to the *macro-averaged* One-vs-One ROC AUC, which is the average of the ROC AUC for each pair of classes and can provide a measurement of the model's ability to discriminate between all the classes.

Bibliography

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of Research and Development* **3** (1959) 210–229.

[2] T. M. Mitchell, "Machine learning", 1997.

[3] OpenAI, "GPT-4 Technical Report", *arXiv e-prints* (Mar., 2023) arXiv:2303.08774, arXiv:2303.08774 [cs.CL].

[4] **CMS** Collaboration, S. Chatrchyan *et al.*, "Observation of a New Boson at a Mass of 125 GeV with the CMS Experiment at the LHC", *Phys. Lett. B* **716** (2012) 30–61, arXiv:1207.7235 [hep-ex].

[5] J. Kiefer and J. Wolfowitz, "Stochastic Estimation of the Maximum of a Regression Function", *The Annals of Mathematical Statistics* **23** (1952) 462 – 466. https://doi.org/10.1214/aoms/1177729392.

[6] C. Bierlich *et al.*, "A comprehensive guide to the physics and usage of PYTHIA 8.3", *SciPost Phys. Codeb.* **2022** (2022) 8, arXiv:2203.11601 [hep-ph].

[7] J. R. Christiansen and P. Z. Skands, "String Formation Beyond Leading Colour", *JHEP* **08** (2015) 003, arXiv:1505.01681 [hep-ph].

[8] **GEANT4** Collaboration, S. Agostinelli *et al.*, "GEANT4–a simulation toolkit", *Nucl. Instrum. Meth. A* **506** (2003) 250–303.

[9] J. H. Friedman, "Greedy function approximation: a gradient boosting machine", *Annals of statistics* (2001) 1189–1232.

[10] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Journal of computer and system sciences* **55** (1997) 119–139.

[11] L. Breiman, *Classification and regression trees*. Routledge, 2017.

[12] J. R. Quinlan, "Induction of decision trees", *Machine learning* **1** (1986) 81–106.

[13] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system", *CoRR* **abs/1603.02754** (2016) , 1603.02754. http://arxiv.org/abs/1603.02754.

[14] B. Kegl, CecileGermain, ChallengeAdmin, ClaireAdam, D. Rousseau, Djabbz, fradav, G. Cowan, Isabelle, and joycenv, "Higgs boson machine learning challenge", 2014. `https://kaggle.com/competitions/higgs-boson`.

[15] R. Mitchell and E. Frank, "Accelerating the xgboost algorithm using gpu computing", *PeerJ Computer Science* **3** (2017) e127.

[16] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

[17] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial", *Frontiers in neurorobotics* **7** (2013) 21.

[18] XGBoost Documentation, "Xgboost parameters." `https://xgboost.readthedocs.io/en/stable/parameter.html`.

[19] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework", in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631. 2019.

[20] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization", *Advances in neural information processing systems* **24** (2011) .

[21] P. I. Frazier, "A tutorial on bayesian optimization", *arXiv preprint arXiv:1807.02811* (2018) .

[22] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms", *Advances in neural information processing systems* **25** (2012) .