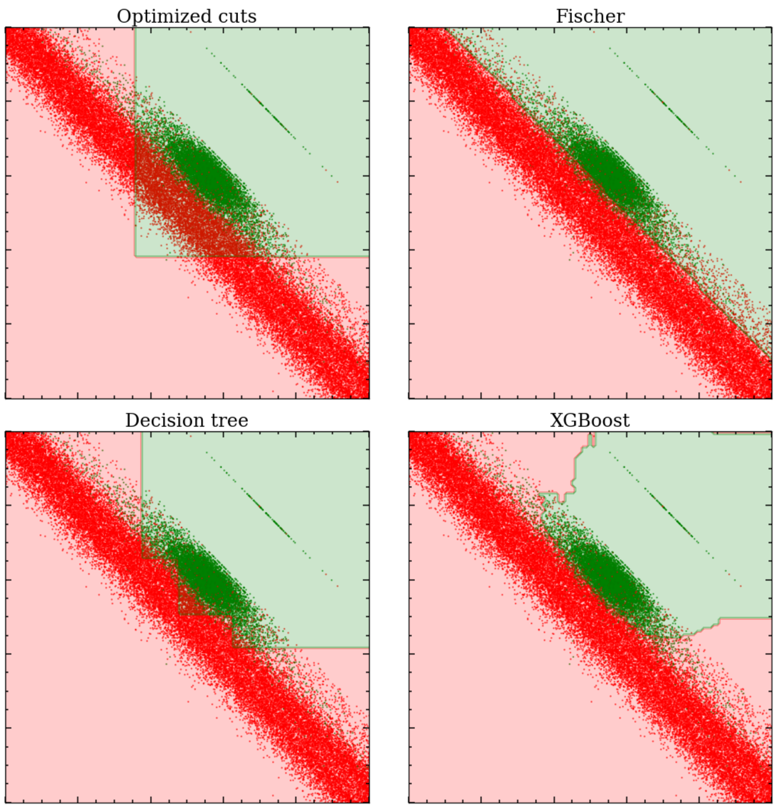
Københavns Universitet: Bachelorstudiet i fysik

Projekt Uden for Kursusregi 2020

Enhancing Physics Research Potential of High Energy Physics Experiments Using Machine Learning Techniques.

Performed on V^0 candidates from LHC Run 2



Authors:

Jonas Vinther	KU- ID: dlk339
Johann Bock Severin	KU- ID: msk377
Jakob Hallundbæk Schauser	KU- ID: pwn274
Christian Kragh Jespersen	KU- ID: htd809

Advisor:

Troels Christian Petersen	Email: petersen@nbi.ku.dk

This report consist of **26** pages of main text and **13** pages of appendices.

The report was handed in the 30th of October, 2020.

Contents

1 Introduction

In many branches of physics, new discoveries are made by quantitatively identifying a signal peak upon a background of noise. Because of the importance that peak finding and classifying has in all branches of physics, these methods will be the main focus of this project. In High Energy Physics, this could be the peak of a new particle found amongst the almost uncountable amount of collisions happening every second the LHC is running. Few collisions actually produce the particles we search for, so finding and quantifying small peaks is the cornerstone of making the actual discovery. Obtaining enormous amounts of data is the bread and butter of the different LHC experiments. In this project we work with data from the ATLAS experiment.

The seemingly endless amount of data is expanded with events from a Monte Carlo (MC) simulation built upon first principles and our understanding of the detector. This all makes the ATLAS data-set of V^0-particles a playground to develop and apply tools for classification of data. The MC simulation provides valuable insight into the training methods possible when the true particle type in simulation is known, as well as providing a framework for validating our procedure.

In this project, the main goal is to explore and test large scale classification tools and evaluate the models in data where the particle type is unknown and the mass peak is the best indicator. The project is structured such that:

- In section 2 the ATLAS experiment, data set and the main features will be presented. This will be followed by a description and comparison of the classification algorithms used in the project.
- In section 3 the analysis and quantification of the data set based on considerations described in the first half of the report will be made. This includes finding the peaks of particles, comparing the Monte Carlo simulation with real data, as well as describing correlations and ranking the importance of the features in the data set.
- The main results of the project will be presented in section 4, where the performance of different algorithms and methods for classification and evaluation will be compared.
- Lastly, the strengths and shortcomings of the methods will be discussed in section 5. This section will also contain suggestions for future work with the project.

When reading the paper, note that:
- All authors have contributed equally to the project.
- If not otherwise stated, we use natural units (e.g. $c = 1$) and when referring to "mass" in the paper, the *invariant* mass is meant.
- We interchangeably use K^0_S/K-Short to refer to the same particle, as well as Λ/Lambda and $\overline{\Lambda}$/Lambda-bar respectively
In general this project did not focus on high energy/particle physics, but rather on how to treat signal-finding in large amounts of data using high-level statistical methods and machine learning. Many of our findings are thus methods that aren't necessarily relating to a clear goal, but investigating the different angles of attacking the problem of finding and quantifying peaks. Whenever possible we have attempted to quantify our findings, since the project focuses on providing a data-driven approach for peak-finding and validation.

2 Data and Theory

In this section the experiment and the data set will be presented. Subsequently, the algorithms of choice and a priori considerations for model evaluation methods will be presented.

2.1 ATLAS Detector Layout and V^0-particles

In the LHC, two protons beams are accelerated to a relativistic mass of $6.5 TeV$. The beams are directed to collide in the middle of the ATLAS detector (or the other detectors in other experiments), which consists of multiple layers of different detectors.

As the purpose of this project is to optimize and apply different algorithms for peak finding and classification, the project aims to analyse data on the physically well understood and frequently occurring V^0-particles. V^0 denotes a heavy neutral particle, that decays into two oppositely charged daughter particles. The name of the V^0 is well-chosen: The shape of the decay resembles a "V" and it has a total charge of 0.
See table 1 for summary statistics of the relevant V^0 particles.
Placing the z-axis of our coordinate system along the collision path, the ATLAS detector is built up of multiple cylinders consisting of different types of detec-

Table 1: Physical properties of considered V^0 particles [1]

	Half-time [s]	Charged Decay	Neutral Decay	Charged/Neutral-ratio [%]	Mass [MeV]
K_S	$(8.954 \pm 0.004) \cdot 10^{-11}$	$\pi^+ + \pi^-$	$\pi^0 + \pi^0$	69.20±0.05	497.611 ± 0.013
Λ	$(2.631 \pm 0.020) \cdot 10^{-10}$	$p + \pi^-$	$n + \pi^0$	63.9 ± 0.5	1115.683 ± 0.006
$\bar{\Lambda}$	$(2.631 \pm 0.020) \cdot 10^{-10}$	$\bar{p} + \pi^+$	$n + \pi^0$	63.9 ± 0.5	1115.683 ± 0.006

tors extending out in the xy-plane. Since the focus of the detector is to measure the momentum from the tracks of charged particles, a $2T$ magnetic field is applied in the xy-plane, making it possible to determine the momenta for charged particles. The ATLAS inner detector consists of three detector types: Pixel (placed at $45.5mm < R < 242mm$), SCT ($255mm < R < 549mm$) and TRT detectors ($554mm < R < 1082mm$), made of 4, 8 and 73 layers respectively. The Pixel detector provides a high resolution detector close to where the protons collide, the interaction point, and is vital for precise tracking. The SCT is similar to the pixel detector but with 'larger' pixels which allows it cover a larger area, and thus compliments the pixel detector in measuring precise hits close to the interaction point. The combination of these precision trackers at small radii with the 4mm TRT straw tubes at a larger radius gives robust and precise track reconstruction. The tracks are described by the following parameters[1]: Transverse momentum (pT), the transverse and longitudinal impact parameters of the tracks d_0 and z_0, as well as the angle ϕ and pseudo-rapidity η[2]. The mass of the particle is not measured directly. It is calculated under a given decay hypothesis, using the track features and covariances.

The objective is then to find the V^0-candidates given these many tracks. To do this two oppositely charged tracks are selected, [3] and the hypothesis that they share a common vertex is tested. If the resulting chi-square is < 15 and the vertex is consistent with that of the proton collision, the candidate is added to the dataset. The track features can now be used to calculate different attributes regarding this candidate along with their uncertainties.[4]
An example of a V^0 reconstruction can, along with the most obvious physical parameters, be seen in figure 1.

[1]And their co-variance matrix

[2]Defined as a transformation of the angle θ with regards to the beam direction: $\eta = -ln[tan(\frac{\theta}{2})]$

[3]The charge is found from the curvature of the track due to the applied magnetic field

[4]A notable background effect from this procedure is that light can create e^-e^+ pairs near a nucleus which would be false positive V^0-candidates. This background is proportional to the detector matter density.

The useful parameters calculated from the vertex are[5]:

- V^0_mass: The calculated rest mass of the V^0-particle assuming it to be a certain type.
- $\cos\theta$: Cosine of the angle between the summed momentum vector of the two decay products and the line from the primary vertex to the V^0-vertex.
- V^0_rxy: The distance from the V^0-vertex to the center of the accelerator.
- $a0\_xy$ and $a0\_z$: The location of the V^0-vertex measured from the primary vertex.
- η: pseudorapidity; the transformation of the angle of the V^0 vertex with respect to the beam (see footnote 2).

The features denoted by pv0_ refers to the attributes of a primary vertex (i.e. the points of proton collision) and can thus contain a lot of duplicates. The features denoted by v0_ refers to the attributes of the tracks originating from the V^0-vertex, except for v0_x/y/z which denotes the location of the V^0-vertex.

As there is a lot of data, throwing away noisy data and unlikely candidates is often better than having a raw, noisy sample. Therefore, cuts can be made to improve the sample, some of which were made before the data was handed to us, since they are made during the reconstruction and are required for the reconstruction to be considered correct.

- $\chi^2 < 15$, as anything higher is too uncertain and might contribute to mislabeling.
- Conservation of charge for V^0-decays implies that the daughter particles be either neutral or of opposite charge. Thus, the reconstruction requires $Q_1 \cdot Q_2 \leq 0$

Other cuts are for convenience and robustness, for example:

[5]The calculation of these from the data collected in ATLAS are complicated and not within the scope of this project

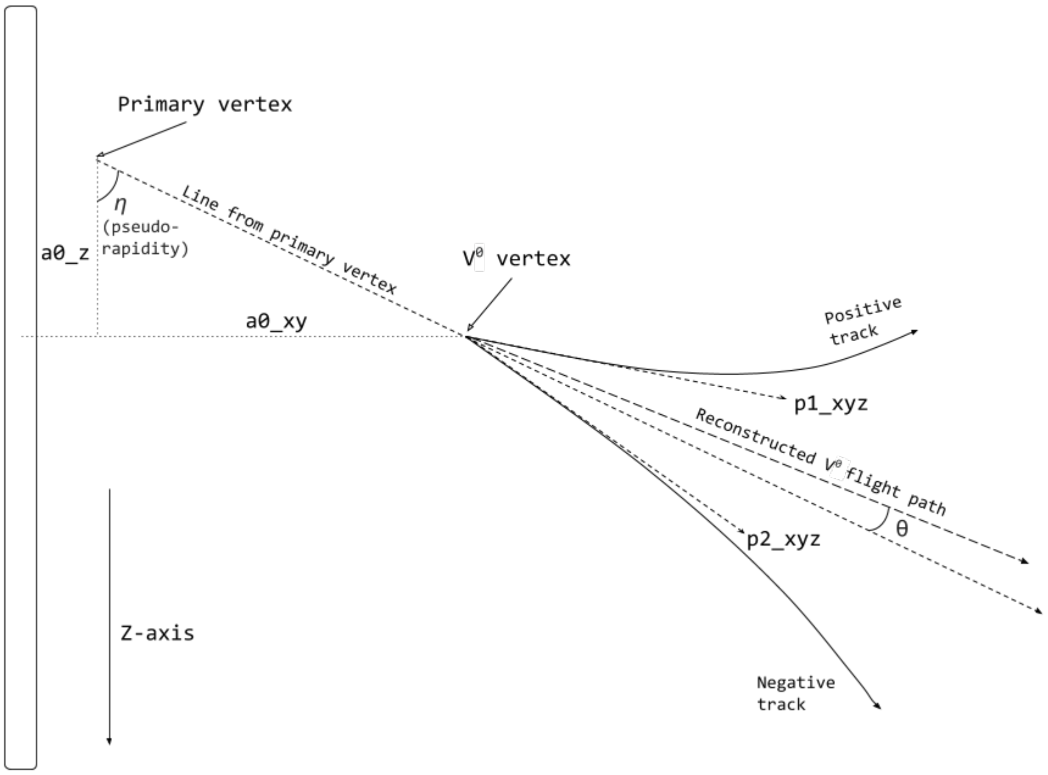


Figure 1: A schematic overview of the calculated parameters from the reconstruction of a V^0-particle decay.

- A good way to improve the robustness would be to force $|\eta| < 2.5$ as the detector layout works best for small rapidities.
- Since most particles do not decay instantaneously, it is also appropriate to require a minimum flight length.

It was quite late that the ATLAS common practice of making these cuts became known to us. Therefore we were not able to implement them globally across the whole project, and it was interesting to implement our own methods. For the sake of consistency we have left them out.

For the main part of this project, the specifics of the decays do not matter, but if further analysis highlights a feature or sets of features, the basic understanding of the physical process serves as a sanity check, and provides further understanding.

2.2 Data and Monte Carlo Simulation

The data used in this project consists of two sets of two different data types: Monte Carlo simulated (MC) data and real data from the ATLAS detector. We generally used only one MC file and one ATLAS data file. The ATLAS data was collected early in LHC run 2. The data used in our analysis is about 2 billionths of the full run 2 data.

The MC simulation is very adept for getting an intuition for finding the signal, because of clear labels of what is definitely signal and what is definitely background. This can provide a valuable resource in analysing the real data. However, real data has many quirks that the MC data set does not accurately portray. This implies some pitfalls when using the simulated data, and makes the raw MC-data distinguishable from the raw real data.

However, the discrepancy between data and MC provides a way of investigating possible new physical discoveries. Since the MC-simulation is simulated from as fundamental a starting point as possible (first principles), and then run through a detection framework as close as possible to the real ATLAS data processing, a disagreement between the MC and real data can have two possible causes:

- We have somehow misunderstood some of the basic principles of physics.
- We do not understand our detector well enough.

While the second is the most likely, smaller corrections of the first kind are also to be expected and can provide valuable insights.

These discrepancies can be circumvented using different methods that will be discussed in section 3.3.

2.3 Classification Algorithms

In this section different types of classification will be covered. To give an idea of the methods described throughout the section, they are implemented in a 2D-space and their decision boundaries are visualized in figure 3.

2.3.1 Simple Cuts

As a simple first step, looking at the Monte Carlo-generated data with the signal highlighted and making cuts by qualitative estimates, gets you surprisingly far.
Most cuts made this way can be rooted in physical intuition. Two examples are;

- The predicted mass of a K_S^0 is around 500 MeV. Particles with masses far from this can be left out.
- Theta is a measure for the angle between the momenta of the decaying particle and its daughter particles. As conservation of momentum requires this angle to be ≈ 0,$\cos\theta$ should never stray far from 1.

However, cutting solely "by hand" might introduce unwanted effects. We tried to optimize the cuts using the minimization algorithm in SciPy[6]. This was done by maximizing the signal significance:

$$\frac{N_{sig}}{\sqrt{N_{sig}+N_{bkgr}}}$$

Since the most desirable outcome is to retain most of the signal while removing the background, we required that a minimum of 95% signal is kept.

The results from this algorithm is given by:

$$\begin{aligned} r_{xy} &< 442.281 \text{ mm} \\ \chi^2 &< 10.617 \\ \cos\theta &> 0.9875 \\ 459.048 \text{ MeV} < m_{K_S^0} &< 546.543\text{MeV} \end{aligned}$$

These are visualized in figure 2.

[6]`https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html`

2.3.2 Fisher

A classic way to make a classification is using the Fisher Discriminant[7]. The simple idea of the Fisher Discriminant is to project points in high dimensional space onto the line that maximizes the distance between the classes compared to the distance within the classes. The maximum separation occurs, with the line given by the vector:

$$\vec{\omega} \propto (\Sigma_s + \Sigma_b)^{-1} (\vec{\mu}_s - \vec{\mu}_b) \qquad (1)$$

where Σ is the in-class co-variance matrix, and $\vec{\mu}_x$ is the vector consisting of the parameter means for class x. The score is now computed by taking the dot-product with the data points, and an appropriate decision line can be chosen ([2]).

2.3.3 Decision Trees

Decision trees are a generalized way of making cuts for prediction. They can be used for both regression and classification but in this project the focus is on the latter. A decision tree consists of numerous nodes (called leaves), each of which divide the input data into subsets which is fed into the next layer of nodes. For classification, the goal is then for the final layer to have nodes which only contain points of a single class. The number of layers is called the "depth" of the tree. When training a decision tree for classification, a measure of impurity for a given node is necessary. Most commonly the Gini-impurity is used, defined by:

$$I_G(p) = \sum_i p_i \left(1 - p_i\right)$$

where the sum is over all possible classes, and p_i is the fraction of points of the class i in the node. An interpretation of the Gini impurity is that if you assigned each point randomly to class i with the probability p_i then the Gini impurity is the fraction of errors you are going to make. Creating a new node usually consists of choosing a cut, which minimizes a weighted sum of the Gini impurities of the two daughter nodes. This is the widely used 'greedy' approach and it should be noted that it does not guarantee the best optimization as it does not account for possible better cuts resulting in an apparent worse earlier cut.

In an effort to not overfit[8], a tree can be pruned [9]. In

[7]with assumptions like normally distributed data this can be generalized to Linear Discriminant Analysis

[8]Overfitting is a term for when a predictor loses generalizability by learning the specifics of the training data instead of general trends

[9]No split is made if the gain is under a certain threshold

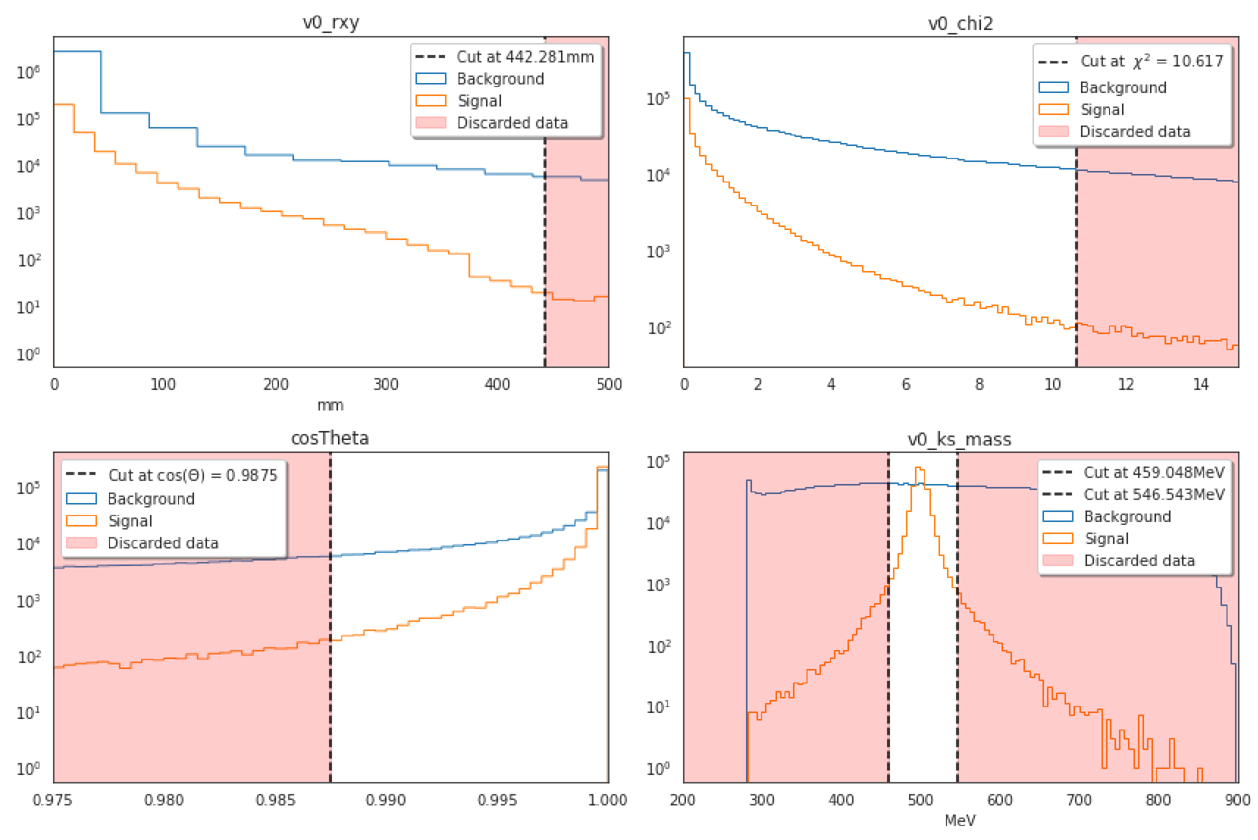


Figure 2: A visualisation the four simple cuts made in **MC**, using truth labels to distinguish signal and background.

the implementations used in this project this is done using the parameter γ, whilst constructing the tree. When enumerating the leafs, the 'gain' is calculated as[10]:

$$Gain_{idea} = [S_L + S_R - S_{L+R}] - \gamma$$

where S_L and S_R are the scores of the new left and right leaves and S_{L+R} is the score on the original. γ is the important constant that controls the regularization. If the gain is negative, the branch terminates, otherwise it is split, and goes on until reaching a set maximum depth [3].

2.3.4 Boosted Decision Trees

While the decision tree ideally separates classes completely, this is rarely a possibility while simultaneously retaining a finite depth and robustness. A better result can often be obtained by producing a lot of 'small' trees and combine them into a forest. This is the procedure of boosting, which denotes the method of combining a group of 'weak' models into a single good model. In the case of trees this is frequently denominated a "forest" [11].

[10]We leave out some prefactors in this formula

[11]This is also called an ensemble model

The forest is built additively a single tree at a time, where each new tree focuses on complementing the forest where the current predictions are weak. This notion can be defined in at least two ways. The first is adaptive boosting where the new tree trains on data which is weighted by how bad they are predicted by the current forest, i.e. a misclassified data point receives a higher weight. The other formulation is as an optimization problem and is called gradient boosting:

If we denote the tree that is currently learning as $f_t(x)$ and our prediction of i at time t as $\hat{y}_i^{(t)}$, the learning process can be described as:

$$\begin{aligned} \hat{y}_i^{(0)} &= 0 \\ \hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i) \end{aligned}$$

The training of the individual tree consist of optimizing the objective-function:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i)$$

Here $l(y_i, \hat{y}_i^{(t)})$ is the loss function to be minimized and y_i and $\hat{y}_i^{(t)}$ is the target and current prediction at time t respectively. Ω is the common notation for a regularization term. The loss function is commonly the Mean Squared Error for regression and negative log-likelihood/cross entropy loss for classification. However, it can be advantageous to expand the loss function in a second order Taylor expansion (with first and second order differentials g_i and h_i as coefficients), in order to generalize the optimization for any loss function. In this case the objective becomes finding the new tree f_t which minimizes:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

The creation of the tree f_t follows a greedy approach since it greatly reduces computational time and doesn't improve forest significantly. [3]

We will now dive deeper into the specific tree-boosting algorithms we have used in this project:

XGBoost There are multiple ways of 'training' the trees. Training means optimizing the tree on the given data and classes. One of the most reliable and efficient algorithms is found in the XGBoost-library (eXtreme Gradient Boosting). As the name suggests, XGBoost implements gradient boosting to quickly produce a large ensemble of weak models.[4]

LightGBM Another very popular algorithm is the Microsoft-created Light Gradient Boosting Machine, better known as LightGBM. The main difference between LightGBM and its competitors (XGBoost for example), is the fact that LightGBM uses a technique called Gradient-based One-Side Sampling (GOSS) for finding the optimal cutting value while XGBoost uses a histogram-based algorithm which is slower in most cases.

After bundling both data and features using GOSS, the algorithm runs as $O(N_{databundles} \cdot N_{featurebundles})$ which trivially beats the $O(N_{data} \cdot N_{features})$ used in other algorithms on larger data sets.[5]

AdaBoost Is an implementation of the adaptive boosting formulation. Adaptive boosting works by switching between training on a section of points and classifying a random sample of the others. Between each iteration, the algorithm re-weights the input space, giving larger importance when training on wrongly classified points. This is consistently slower than the two others, but has the potential to beat the other algorithms on precision in the long run.[12] All of the methods discussed in this section somehow make a decision boundary in the given parameter space with differing levels of sophistication which is displayed for comparison in figure 3.

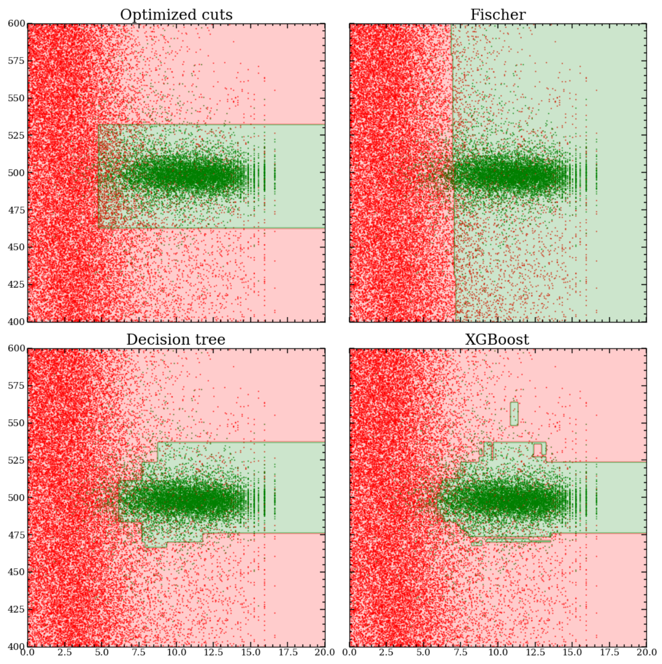


Figure 3: Decision Boundaries in a 2 dimensional parameter space, with $m_{K_s^0}$ on the y-axis and logit$((\cos\theta + 1)/2)$ on the x-axis, in **Monte Carlo** simulation. The decision boundaries display where in the feature space the given algorithm would classify points as signal or background. The points are a test sample to demonstrate the accuracy. The cut in the Fisher Discriminant is found by minimizing the Gini impurity. To see the decision boundaries for two correlated features see figure 38

2.4 Investigative Algorithms

Classification tasks are rarely as simple as just applying one of the aforementioned algorithms. Much of the work goes into preprocessing the data and optimising the chosen algorithm. In this section, we introduce methods used to find correlations, evaluate model performance and quantify the importance of any feature.

2.4.1 Shapley Additive Explanation (SHAP) Values

Normally, when we train a simple model (decision tree, linear regression, etc.) the model explains the result itself. In a linear regression, we can interpret

[12] the implementation used is found at: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html`

the results as a slope and intercept. In a single decision tree the cuts can be written explicitly. However, when we have more complex models like neural networks or boosted decision trees, the results are harder to interpret as the models are more complex. To enlighten us, an approach is to quantify the importance of each input-parameter in the model. To do this we are using the SHAP (SHapley Additive exPlanations) algorithm. [6].

SHAP values are built on the game theoretical concept of Shapley values. A Shapley value is a way to determine the reward an actor in a game should receive according to their contribution. It is calculated for each feature i by predicting a model f using the feature space S without i and a feature space $S \cup \{i\}$ which includes the feature i. The Shapley value is now calculated by summing the contributions from including i, weighted appropriately.

$$\phi_i = \sum_{S \subseteq F \backslash \{i\}} \frac{|S|!(|F|-|S|-1)!}{|F|!} \left(f_{S\cup\{i\}}(x_{S\cup\{i\}}) - f_S(x_S) \right)$$

Where the sum is over all subsets that does not include the parameter i. In the actual implementations several assumptions are made.
Since the models described in section 2.3 cannot handle randomly missing inputs, $f_s(x_s)$ are approximated as the expectation value where the features of S are fixed: $f_s(x_s) \approx E[f(x)|x_s]$. Furthermore, the actual value is found by sampling, as the possible number of permutations of the group $S \subseteq F\backslash\{i\}$ quickly becomes large.

The resulting SHAP-values can be used to see how the parameters affects the classification. The absolute SHAP-value is an indicator of the importance of the feature, and the sign can be used to analyse how the features contribute to the classification. Higher values of $\cos\theta$ gives high positive SHAP-values, as it is a good indicator for signal, as an example.

Most models include some kind of feature importance estimation, but SHAP-values are robust, generalizable and easy to understand.

2.4.2 Correlations and Maximal Information Coefficient

In this project, when doing feature selections, minimizing correlation with mass is vital. A normal approach for quantifying correlation would be to use the Pearson correlation coefficient defined by:

$$\rho_{x,y} = \frac{\mathrm{cov}(x,y)}{\sigma_x \sigma_y}$$

This method is computationally efficient and is good at finding linear correlations. Thus it would have been a suitable approach had we used a linear classifier such as Linear Discriminant Analysis.

However, the primary method used for classifying is boosted decision trees, which are far from linear models, and are therefore sensitive to other types of correlations. To calculate more general correlations we use the Maximal Information Coefficient (MIC) [7].

This correlation type builds on the measure of mutual information, which is defined by:

$$I(X,Y) = \int dxdy\; p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)} \qquad (2)$$

This compares the probability distribution over two variables. A more illuminating form is to write out the logarithm: $\log_2 p(x,y) - \log_2 p(x)p(y)$ which can be seen as the bitwise difference in the information contained in the joint probability between x and y compared to the assumption that x and y are independent variables[13]. Computationally the mutual information $I(X,Y)$ is estimated by binning the data. Binning over X, Y bins gives:

$$I(X,Y) \approx \sum_X \sum_Y p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}$$

However, this measure is quite sensitive to the choice of binning. To make up for this we use the MIC-algorithm. Which maximizes:

$$\mathrm{MIC}(X,Y) = \max_{|X||Y|<B} \frac{I(X,Y)}{\min(|X|,|Y|)}$$

The mutual information is thus maximized for different binning schemes of X and Y that satisfy having sizes less than some number B. The use of MIC to determine correlation now gives us a general correlation metric that could be used for very differently shaped probability distributions. This is however computationally heavy, and we have to sacrifice the amount of data used to determine the correlation (to a practical maximum of $\approx$ 5000 events).

[13] $p(x,y) = p(x)p(y)$ if and only if x and y are independent variables

2.4.3 Receiver Operating Characteristic (ROC)

For evaluating the perfomance of a model, a lot of different metrics are available. One of the simplest is the Receiver Operating Characteristic (ROC), which is a graph displaying the relationship between the true and false positive rate of classification. Thus, given the prediction scores of a model, the ROC-curve shows the relationship between correctly classified signal and background events for different thresholds in the prediction scores.
The rate at which these are classified are denominated as TPR (True Positive Rate) and FPR (False Positive Rate). These can also be denominated as signal- or background efficiency.
Given a dummy classifier assigning scores at random, the ROC-curve would be along the diagonal, as the rate of truthful classification would follow the ratio of signal and background events in the data. With a perfect model, the ROC-curve would go straight to the corner, as there would be only background up to a given score from the model, and the rest would be signal. This form motivates the use of a metric for determining the usefulness of a classifier by comparing the integral of the ROC-curve, the AUC (Area Under Curve), which would be 0.5 for a dummy classifier and 1 for a perfect one.[14]

ROC-curve by fitting Usually the true/false positive rate is calculated by the use of truth-labels. However, whenever it is possible to estimate the true/false positive rate, it is also possible to estimate the ROC curve. In this data-set, using the mass histogram, we are able to estimate the ratio of the amount of signal to the maximal amount of and the ratio of the amount background retained to the maximum amount of background given some classification. To do this, we leverage the fact that the signal can be fitted as a localized peak on top of the background.

The procedure starts by finding the total amount of signal and background in the test sample by fitting the mass histogram. Thereafter, cuts in the prediction scores are made and once again the mass histogram is fitted to find out how much signal and background is retained. The procedure can be summarized as the following:

[14]If the AUC is below 0.5 it can be inverted to still give a predictive power over random, but this report does not include those instances.

- Fit the entire test sample to find the maximum amount of signal and background.
- Determine a threshold in the prediction scores, leaving data points with scores beyond the threshold as a subsample.
- Find the amount of signal and background left in the subsample through fitting.
- Convert this to ratios between the maximum amount of signal and background respectively, which gives the TPR and FPR respectively.
- Move the threshold and repeat.

This procedure works best if the mass histogram does not fluctuate much for different thresholds. For example, the background should be removed 'uniformly', especially around the signal peak. For example, if, at some threshold, the background just outside the peak is removed but the background under the peak is kept, one could erroneously achieve a signal ratio larger than 1.

Another pitfall when using this method at low-signal instances can the background and peak fit erroneously make a small negative peak. This leads to small fluctuations around x ≈ 0 when drawing the ROC-curve based on fitting.

ROC-curve by cross-validation A more traditional way of making a ROC-curve is making two models where the probability scores are then used to cross validate each other. The procedure then goes as;

- Train two models on two uncorrelated sets of parameters, and retrieve probability scores from each.
- Use the probability score from model 1 as a truth label and evaluate the ROC-curve for model 2 (or vice versa).

This can be employed in a variety of ways, of which two will be implemented in this project:

- Cross-validating two models trained on separate uncorrelated parameters in data.
- Cross-validating a model trained on a fuller set of parameters in data, with a model trained on the same parameters in MC. These two are heavily correlated, so the main takeaway from this kind of validation would be to check if the two

models extract the same amount of information from the data.

It should be noted that a given model should only be cross-validated against a better model, since the worse model will not be susceptible to false positives in the pseudo-labels. However, this also means that cross-validation ROC-curves can be a tool in telling which of two models is the better predictor, ie. the better model would pick up on the false positives provided by the worse model.

3 Analysis

In this section the initial analysis of the data will be presented. First, the peaks for K-Short and Lambda particles will be quantified. Then the features of MC will be compared with the data. Afterwards the correlation of features with the mass as well as with each other will be determined. In the end of the section, the importance of the features will be ranked and the ROC-curve method suggested in section 2.4.3 will be tested in MC.

3.1 Fit and Estimates

In this section we will discuss the peaks associated with the particles.

3.1.1 K_S^0 (K-Short)

A good first approach for describing the K_S^0 is fitting the mass peak. As the peak is made from events with different mass uncertainties, a single normal distribution is not a good fitting function. The preferred fit in this project is a double-Gaussian with a single mean μ. To minimize the correlation between fit parameters, the heights of the two Gaussians are controlled by two parameters, a size N as well a fraction f, where the size of the two Gaussians are fN and $(1-f)N$. The peak is found on a background, for which a simple third degree polynomium is used. An example of a background-fit and subsequent peak-fit in both MC and data can be seen in figure 4.

A way of controlling the entire framework of track reconstruction and mass calculation is to check if the mass-error is consistent with the peaks found in the data. This can be checked by picking a number of bands in binned 'v0_ks_massErr' data, and for each one try a single Gaussian fit in the K_S^0-mass was performed. The standard deviation on the fitted Gaussians as a function of the given mass uncertainties can be seen in figure 5. A linear fit between the found uncertainties and the given values has a p-value of 0.219 giving us no reason to suspect any internal inconsistencies. We did, however, find a finite y-intercept, and seeing as the predicted connection between the two should be a direct proportionality with no y-intercept, there might be an error in our methods, either when fitting or binning. But as a systematic error might be to blame, we will try to keep this $\approx 0.765\sigma$ discrepancy in mind.

3.1.2 $\Lambda/\overline{\Lambda}$ (Lambda/Lambda-bar)

The peaks in $\Lambda/\overline{\Lambda}$ are less pronounced than the peak for the K_S^0. The procedures and methods used in order to isolate the peak were the same as for the K_S^0. Only the background fitting required additional tuning, since the background shapes are different. The initial fit results can be seen in figure 4.

3.2 Parameters and Correlation

A lot of work has been done to determine a viable set of features to use for classification. As the complexity in a machine learning model scales heavily with the amount of features, there is a big performance gain when low impact features can be thrown out. Moreover, when we are defining our pseudo-labels for a machine learning model in data, we define it based on mass. This means, that a model correlated with mass will be able to determine whether an event has the appropriate mass for the wrong reason.

3.2.1 Correlations with Mass

To determine the correlations with mass, we use the MIC method described in section 2.4.2. The calculated MIC along with Pearson correlation for data and simulation respectively for the most correlated features can be found in table 2.

As we use pseudo-labels defined from the mass, the correlated parameters makes it possible for a boosted decision tree to determine the mass, and it will thus classify based on this instead of other useful parameters from the peak. To illustrate this example the 10 parameters from the table 2 is used in a LightGBM classifier with results displayed in figure 6. In this fig-

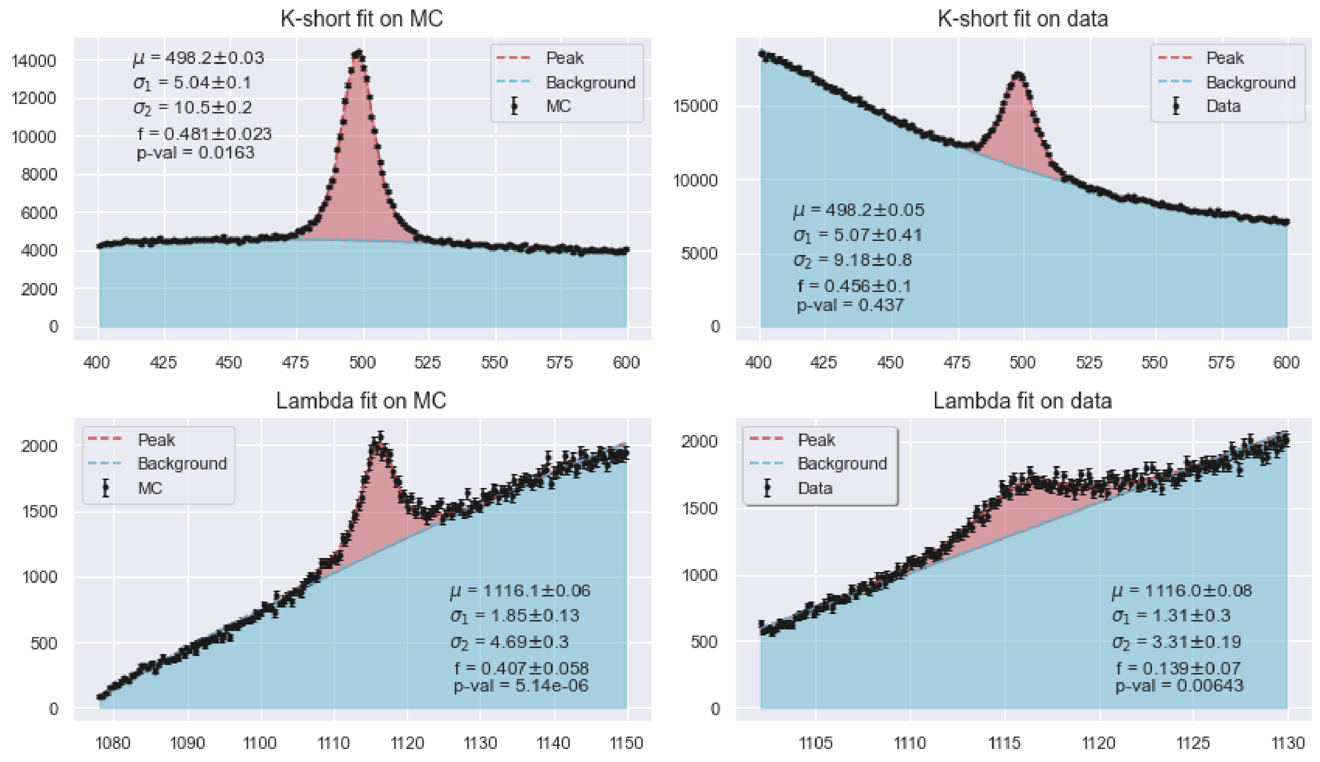


Figure 4: A typical mass-fit in untouched **MC** and **data** on both K-Shorts and Lambda. The background is fitted with a third degree polynomial and a double Gaussian is used on the peak. The parameters μ, σ_1 and σ_2 are what you would expect. f is the fraction of the scale each of the two Gaussians gets (see section 3.1).

Table 2: Table displaying correlation with mass both using the MIC method as well as the linear correlation using the Pearson coefficient

Feature	MIC (Data)	ρ (Data)	MIC (MC)	ρ (MC)
pT	0.47	0.68	0.47	0.67
v0_ks_massErr	0.38	0.49	0.34	0.57
alpha/Alpha	0.28	-0.02	0.26	-0.07
calpha	0.22	-0.38	0.21	-0.38
v0_rxyErr	0.17	-0.20	0.18	0.19
cosTheta	0.15	-0.03	0.18	-0.05
pL1	0.13	0.13	0.12	0.05
v0_rxy	0.13	-0.14	0.12	-0.10
v0_thetastar	0.13	0.01	0.13	0.08

ure, the peak is significant, and is much greater that the one seen before a classification. The background is clearly not a consistent form, so the model classified based on mass.

Some of the parameters are very strong for classification like $\cos\theta$ and r_{xy}. Evaluating their MIC with mass, we get MIC $\approx$ 0.15 for both, which in the MinePy documentation[15] is resembling something between a random Gaussian (MIC of 0.1) and a slightly skewed one (0.2). Along with a small Pearson correlation, this gives no reason to exclude the variables. We note that one should be careful if a classification even vaguely resembles the one shown in figure 6.

After having refined the methods on the K^0_S, we ran the same analysis on Λ and $\bar{\Lambda}$. The correlations were generally stronger in these cases. While some of the features correlate with both K^0_S-mass and the Λ-mass, especially thetastar, v0_p2, v0_qOverP2 and PL2 stands out as more correlated with the mass of the Λ. The exact correlations can be seen in table 8 in the appendix.

3.2.2 Correlations between Features

To investigate and reduce the number of variables, variables were grouped based on the correlations calculated by the MIC-score. Using a data set of 1000 events[16] and taking the first 57 features (those are the mostly continuous features that we chose to include) we now computed the pairwise MIC score and saved them in a correlation matrix.

To group the parameters, the matrix should be set on a block diagonal form. This is done using hierarchical clustering which takes a distance matrix and iteratively merges points given a criteria (here minimizing the variance of a given group). For this clustering

[15]see `https://minepy.readthedocs.io/en/latest/index.html`

[16]Ideally this would be a much larger number. But as we compare $\approx$ 50 parameters pairwise with the computationally expensive MIC algorithm, this is our chosen limit

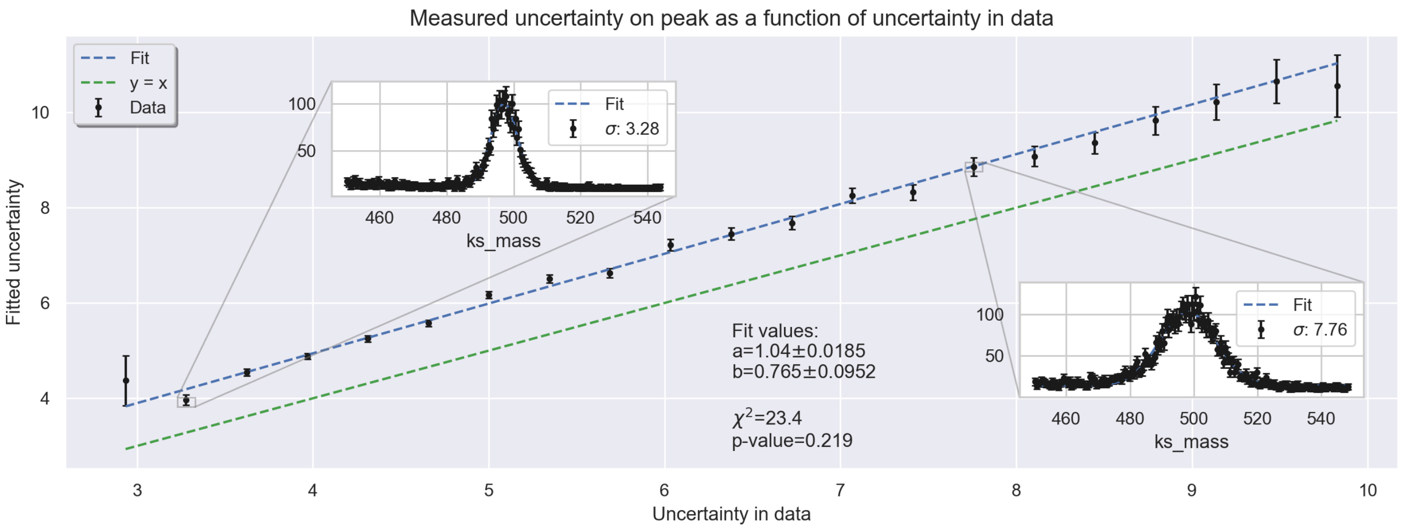


Figure 5: The correlation between the calculated uncertainty in **data** and the uncertainty found through fitting a Gaussian to the same data. The results has been fitted with a line. The fit for each point can be seen in figure 40 in the Appendix.

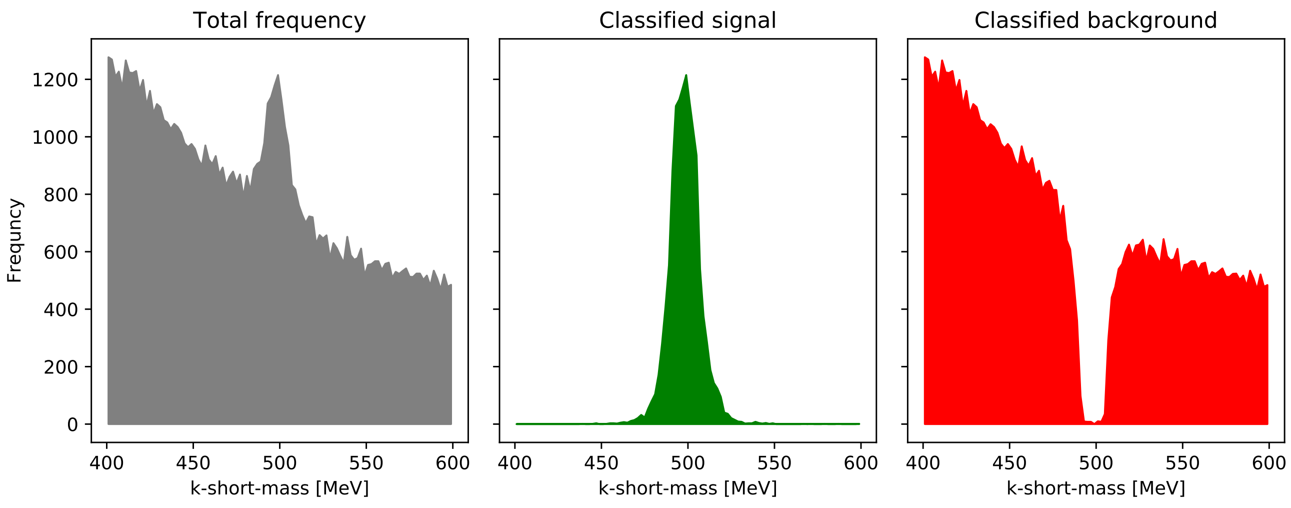


Figure 6: What not to do! Test sample in data with mass between 400-600 MeV. The data is seperated using a LightGBM trained with correlated features displayed in the Table 2. It is very clear that LightGBM misclassifies much of the background purely based on the particle mass.

the distance was defined as $||x - y|| = 1 - \text{MIC}_{xy}$. Keeping track of the merging order, the optimal ordering can be found, which is then used as the optimal ordering of the matrix. Using the hierarchical clustering implementation in SciPy[17], the clustering gives the correlation matrix shown in figure 7.

This figure shows a clear grouping of the features into separate groups consisting of physically similar properties. The groups can be interpreted as (going from upper left to lower right along the diagonal:

- Primary vertex positions
- Calculated features of the V^0-vertex position along with the angles and curvature of the two daughter particles paths
- Momenta for the two daughter particles
- A group of flight length features and $\cos\theta$
- χ^2 in a group with itself

When combining the information from figure 7 with the correlation with the mass, it becomes apparent that the second group (epsilon1-pT) contains multiple parameters correlated with the mass (pT, alpha, pL1, etc.). Since these parameters are evenly spread across this group it would be very difficult to use parameters from this group without correlating the model

[17]Implementation found at `https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html`

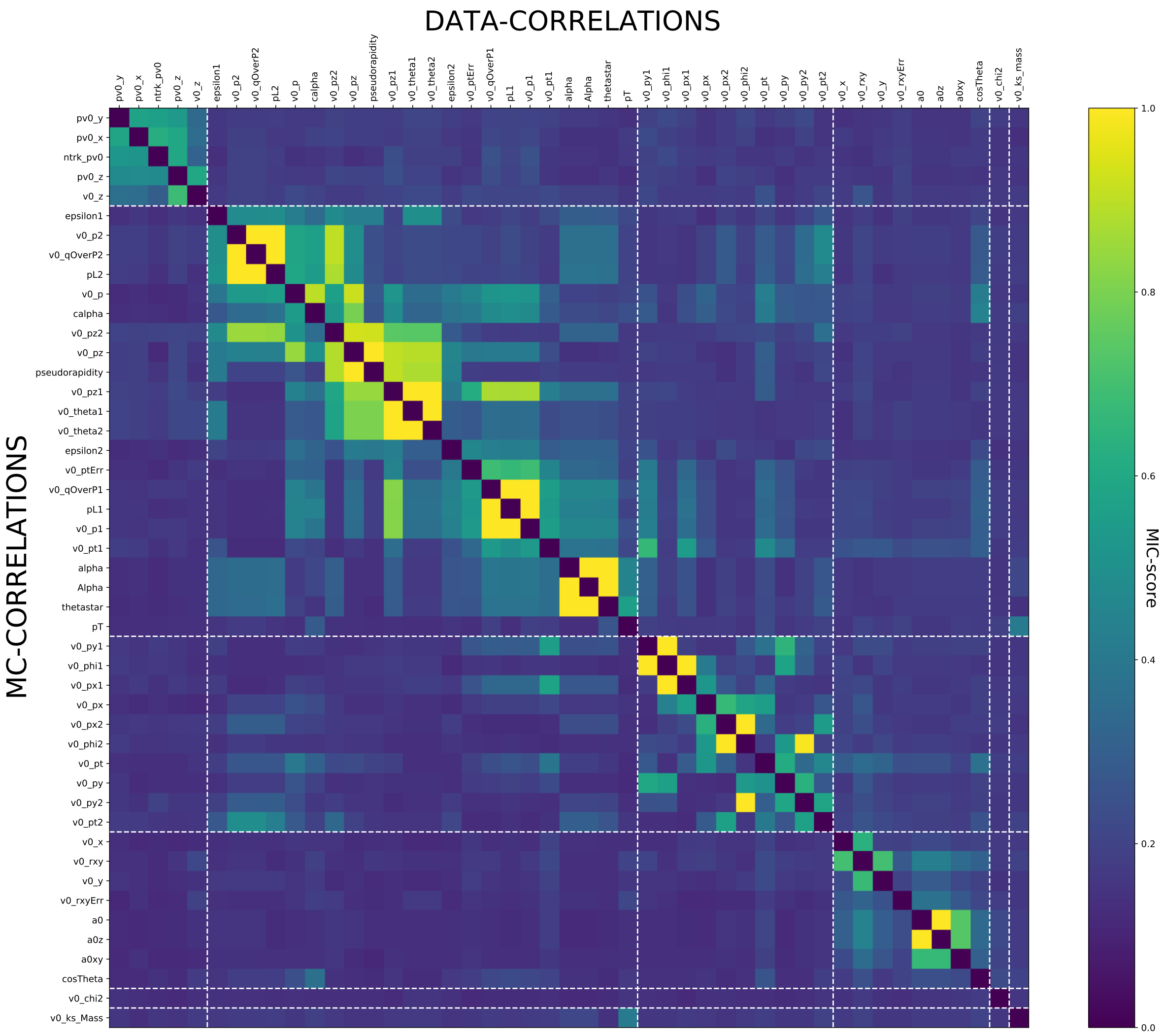


Figure 7: The correlation matrix found by calculating the Maximal Information Coefficient between the features of the data set. The upper right triangle is displaying correlations in **data**, and the lower left triangle is correlations in the **Monte Carlo** simulation. All correlations are calculated using a 1000 point sample. The order of the features are found by doing a hierarchical clustering to find the optimal ordering, and the white lines are separating the groups of features using maxclust = 5 argument. The mass was excluded from the clustering but is added in the end to display correlations.

with the mass. As the main goal is to obtain a smaller sample of features where at least two groups are uncorrelated, it would make sense to exclude the entire second group. This motivates defining two sets of variables uncorrelated with mass defined by:

ML1 features: v0_chi2, v0_px1, v0_phi1, v0_py1, v0_py, v0_py2, v0_phi2, v0_px2, v0_px
ML2 features: cosTheta, a0xy, a0, v0_y, v0_x, v0_rxy v0_rxyErr, v0_z, pv0_z, pv0_y, ntrk_pv0, pv0_x

We also define a third group, **ML1+2** as the junction of the two.

3.2.3 Feature Importance

In this section, the importance of the features in classifying K-Short particle will be quantified. This is done using the features kept from ML1 and ML2 defined in the above section and calculating the SHAP value. Furthermore, the XGBoost implementation also has a way of ranking feature importance using a gain criterion. This value is calculated by seeing the improvement of the splits when they include a certain parameter in a split, and is thus an indicator for the most important values in training the model[18]. The results from calculating the SHAP-values and Feature Importance Scores (with the gain-criterion) can be found in table 3.

Table 3: Calculated absolute SHAP values from an XGBoost model trained on **data**. The SHAP values are calculated from 10^4 points, and the feature importance is calculated by XGBoost according to the gain.

Feature	SHAP-val	Feature-importance
a0	22.7	9.9
cosThetas	19.6	56.3
v0_py1	19.1	3.0
v0_px2	18.1	3.1
v0_py2	16.6	3.2
v0_rxyErr	16.5	3.8
v0_px1	16.2	3.3
v0_z	14.4	1.1
v0_rxy	9.2	3.7

Clearly, $\cos\theta$ is by far the most important parameter for training this model, but not necessarily the most important according to the calculated SHAP-values. Luckily, the SHAP-values are calculated for each point, so it is possible to investigate different SHAP-values more closely. Making a summary plot of the SHAP-values calculated in order to rank them in table 3 the plot in figure 8 is obtained. This figure gives a more nuanced picture of the impact of variables. Here, high values $\cos\theta$ is associated with a more probable signal. For this reason a single split in a decision tree reduces the impurity in the sample significantly and will be a natural choice of a greedy algorithm resulting in higher "feature importance" of $\cos\theta$. The flight length variables a_0, r_{xy} requires multiple cuts to be able to determine signal placing them further down in the decision trees. The gain in purity is drastically decreased by these cuts but it is still vital to the final classification.

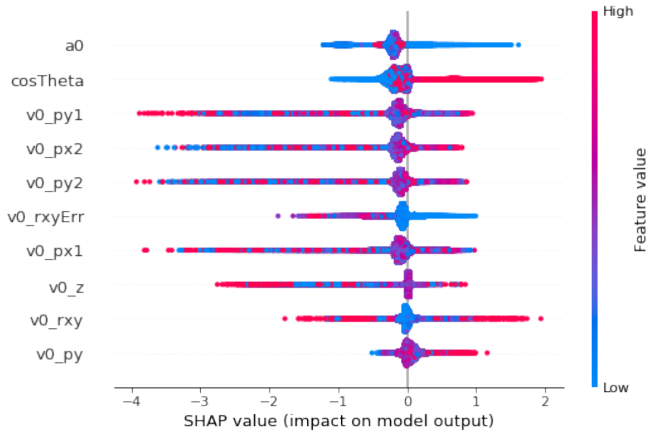


Figure 8: Summary plot of the SHAP values for one group of variables in **data**. A higher SHAP value implies that the feature increases the chance that the point is classified as signal. The higher the value, the bigger impact.

3.3 Comparison of Data and Monte Carlo

In the following section, if nothing else is given, we reference the K^0_S fitting.

The main difference between real data and Monte Carlo (MC) simulated data is that MC has a clear truth label. This means supervised learning methods can easily be trained on MC and applied on real data. However, this approach assumes that the feature spaces of data and MC are indistinguishable, which is rarely the case. First principles simulations are fantastic, but many hidden effects in the detector setup (or wrong simulation principles) make any variable distribution differ substantially between MC and real data. Often there are effects available in the distributions which makes them distinguishable. Some of the effects are:

- Different value ranges. Either the ranges do not overlap, or one of the ranges are smaller, which

[18]found here: `https://xgboost.readthedocs.io/en/latest/R-package/discoverYourData.html?highlight=feature%20importance`

means points outside this range are sure to belong to the other group.

- Dissimilar distribution shapes. Thus, in the higher dimensional feature space, it is possible to distinguish points belonging to either data or MC.

One can try to scale and move the distributions such that they overlap as much as possible, but this is a highly non-trivial task, which will be discussed in subsection 3.3.1.

One clear way to quantify how indistinguishable the feature space of data and MC are, is to train a model to predict whether a point is from a data sample or a MC sample. The AUC of this predictor should then be ≈ 0.5, if MC is indistinguishable from data.
Another way, which is more investigative, is to compare the distribution of data and MC in each feature. However, since we ultimately want to train a predictor on these distributions we need to compare the underlying labeled distributions, i.e. compare signal in data to signal in MC and likewise for the background.

Extracting the signal distributions from MC is trivial due to the given truth labels. We can train a model to estimate the truth labels in data, or we can estimate the signal distributions from the data alone. The procedure of the latter is shown in figure 9. By the use of the peak in mass we select a subset of points which we know to contain signal and background, and through fitting we estimate what the signal to background ratio is. Then we randomly select enough points from the side bands to subtract all the background from the signal/background distribution as shown to the right in figure 9. This procedure works best if the feature is uncorrelated with mass or if the sum of the distributions from the side bands compare well to the background distribution from the signal and background band. In figure 33 in the Appendix we compare the estimated signal distribution against the true signal distribution by applying the procedure in MC.

In the appendix figure 27, we show a gross overview of the comparison between MC and data distributions for 20 different features. We see cases where the range of values are not overlapping, such as pv0_y and pv0_x, or cases where the limits of the ranges are different, such as v0_py/px where MC extends further out than data and v0_y/x where data extends further out than MC. However, from the fractional comparison we see that within the overlapping regions, they are generally comparable, and a solution for the general case is to shift and scale the distributions.

If the distributions are experiencing dissimilar shapes a solution can be to reweight the simulated data.

3.3.1 Reweighting

The idea of reweighting a distribution to match a target distribution is very valuable when comparing a simulated distribution to a distribution with real data. The idea assumes that two distributions span the same values in feature space, but have different densities, leading to a mismatch between the two. This affects training a model on one distribution and applying it on another, since, in practice, the training of most models are based on statistical learning, which is susceptible to feature space density differences.

Thus, ideally there is a high-dimensional analytical mapping that makes the original distribution similar to the target distribution by assigning weights to each data point.

The reader should note that reweighting cannot account for a difference in value ranges but only for the distribution density in a given feature space.

One simple way this can make a difference is when one is trying to distinguish between two classes A and B, but in the simulated data one of the classes is over-represented compared to the test data. Here one could use "zeroth-order" reweighting and simply reweight the amount of class A and the amount of class B to match between the distributions.

We will attempt to perform this task without any fitting or classification of the data. We do this by using a gradient boosted decision tree to approximate the assumed high-dimensional mapping mentioned above.

There are many possible problems with this, most prominently, the curse of dimensionality and discontinuous distributions.

The first problem, high dimensionality, is an issue due to distances in high-dimensional space. This means that a point, although not an extreme outlier in any one distribution, can suddenly appear to be so extreme that its weight is adjusted in a likewise extreme way. However, this pitfall can be avoided by e.g. comparing SHAP values (see sec 2.4.1), in order to remove variables with little impact, lowering the fitting dimensionality and lowering the probability of "misweighting".

Secondly, the discontinuous distributions violate the

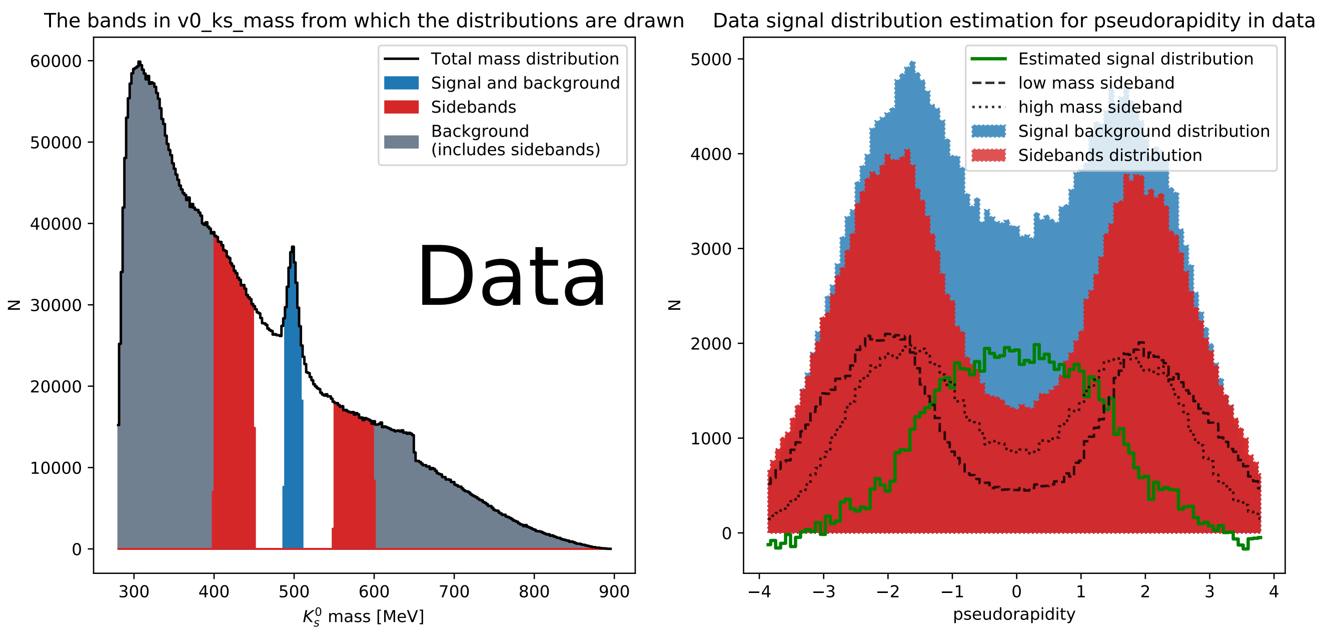


Figure 9: To the left, the definitions of the sidebands and the signal/background-band are shown for 5881544 V^0 candidates. The signal to noise ratio in the s/b-band is estimated through fitting, such that the s/b-band is found to contain ~236e3 background points and ~91e3 signal points. Half the number of background points are randomly selected from each side band in order to draw the distributions shown to the right, where all the points from the s/b-band are shown as the blue distribution. Here the distributions are drawn for the feature pseudorapidity, and as can be seen, the signal distribution is very different from the background distribution, which has larger contributions at high $|\eta|$, where the detector has lower performance.

essential assumption behind reweighting, unless the original and target distributions are discontinuous in exactly the same way. This can be mitigated by transforming the data before reweighting, but in some cases, finding a useful transformation is highly non-trivial.

We attempt to do this transformation for a general dataset by making a scaler that is robust against outliers, placing most (70%-90%) of a distribution in the interval [0,1]. This accounts for the discrepancies in distribution ranges, and often for outliers. However, we have learned that when the number of datapoints eclipses 1 million[19], the reweighting generally fails for sets of variables that includes any variable with a discontinuous distribution, i.e. data and MC remain distinguishable or the weights diverge. Additional to using a classifier to check the distinguishability between MC and data, we perform Kolmogorov-Smirnov 2-sample tests across the distributions.

In both cases, doing manual scaling before doing the reweighting is more effective, since it eliminates any obvious cuts that can be used to distinguish between the two.

[19]Note that it is the implementation that fails numerically, there is nothing that inherently should make this worse when raising the amount of data points

The importance of this can be seen especially in the differences in the discontinuous variables, 'pv0_x', 'pv0_y', pv0_z' and 'ntrk_pv0' as in figures 41a and 41b.

After the scaling and reweighting the average KS-value (defined as the sum of KS two sample tests divided by the number of tests), became ≈ 8 times lower.

The reader should note that in some instances, the discontinuous variables can be left out of the reweighting, and still enjoy a successful reweighting, applying weights trained on the remaining variables.

It is also important to reweight the distributions with respect to the target variable, which in this case was the mass, since any reweighting should not make the target distribution significantly worse. In our code, this was implemented as requiring the KS two sample test value of the mass distribution to improve when reweighting.

3.4 Training in Monte Carlo/Data and Classifying Real Data

The goal of this section is to describe how to obtain a model which successfully can predict the particle type of a given real observation.

Training a classifier in MC is straightforward since truth labels are provided. However, it is not certain a model trained on MC generalizes very well to real data.

At least two approaches can be made to obtain a more accurate model, both of which have been described above.

1. Scale the input features. This first step improves the application of the MC trained model in data, if there is a discrepancy between the feature spaces.

2. Adjust the distributions in MC to become more like the distributions in data, i.e. reweighting. As explained in section 3.3.1 reweighting means adjusting the density of points in feature space.

Training in data is not always a viable option. The reason it is possible for this dataset is the same reason given in section 2.4.3. However, our truth labels are no longer exact, so we define so called pseudo-labels which are estimated truth labels that contain false positives. The pseudo-labels for training data are defined very analogous to figure 9 where points under the peak in mass are labelled as signal and points in the side bands are labelled as background. This means the truth labels are very noisy, but we have found XGBoost to be robust against noisy labels (see section 3.5).

One can also define pseudo-labels by training a model in MC and predict pseudo-labels in data for a data trained model. This method shows little improvement in evaluating the K-Short peak, but would supposedly be adept for a model where no clear pseudo-label can be assigned in data for any parameter, given that the feature space of the data can be made sufficiently similar to the simulated feature space, as discussed above.

3.5 Model Evaluation

In this section, it will be verified whether the suggested ROC-curve methods of section 2.4.3 are actually valid. To test this we will use the previous findings from the correlation with mass from section 3.2.1

The two sets of features defined as ML1 and ML2 are now used to train two boosted decision trees (here the XGBoost implementation) on the pseudo-labels generated from the mass with the method described in section 3.4. As Monte Carlo is used, the truth labels are available, and it is now possible to draw the ROC-curve with both the use of the method described in section 2.4.3 and with the help of the actual label. The method along with true ROC-curve are shown for the ML1 and ML2 model in figure 10. It is seen that in both models the method developed for finding ROC-curves are in agreement with the truth label. The same can be said about the AUC-scores that in these examples are off by ≈ 0.01.

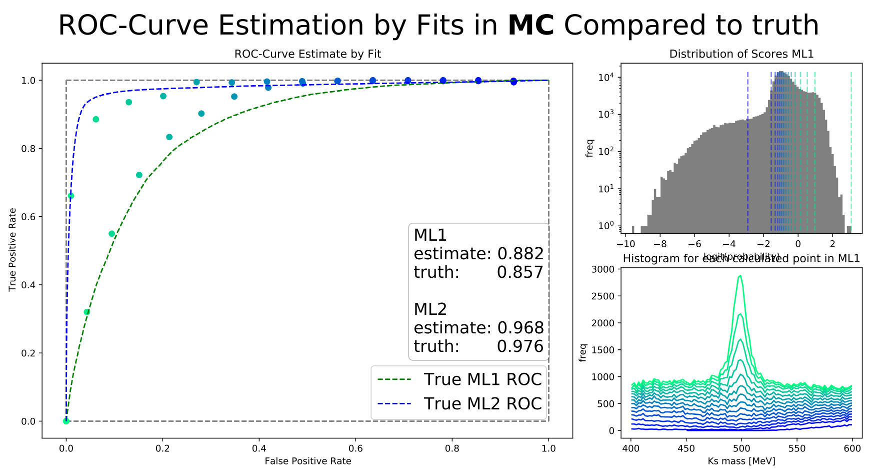


Figure 10: Comparison of ROC-curves on ML1 and ML2 trained on pseudo-labels in Monte Carlo. The ROC curves are made by applying cuts in the probability score (upper right figure and fitting the resulting histogram distribution of mass (the plots in lower right figure). The ROC-curves for ML1 and ML2 are drawn in left figure along with the ROC-curves generated from the truth-label which is accessible in MC.

4 Results

In this section the main results from the project will be presented along with key figures. The results will first be described with the K_S^0-particle where robustness and effectiveness of the algorithms are checked against a significant peak. Afterward, the generalizability of the methods will be tested against another less apparent peak. Here the significantly smaller and more difficult peak of the Lambda-particle is used.

4.1 K-Short (K_S^0)

In this section we will present the results of the algorithms when we apply them on the K-Short particle. Firstly, the K-Short peak is fitted, as described in section 3.1 by a double Gaussian with a fixed mean value. For a sample size of $1.2 \cdot 10^7$, the mean was found to be for the peak at $\mu = (498.2 \pm 0.03)$MeV and a double Gaussian fit with variances described in table 4. We will in this section be using the two sets of features ML1, ML2, from section 3.5.

	K-Short	Lambda	Lambda-bar
μ [MeV]	498.2±0.03	1116.1±0.08	1116.0±0.09
σ_1 [MeV]	5.07±0.41	1.85±0.13	1.31±0.3
σ_2 [MeV]	9.18±0.8	4.69±0.3	3.31±0.19
f	0.456±0.1	0.407±0.06	0.139±0.07

Table 4: The found values for the three particles on undisturbed **data**. f is the fractional height of the two Gaussians.

In figure 11[20] we preliminarily show the results of training XGBoost models with ML1 and ML2 features, on their respective halves of $4.05 \cdot 10^5$ real data events with pseudolabels.

4.1.1 Classical Classification Models

However, before diving deep into the world of machine learning with the entire arsenal of boosted decision trees it can be beneficial to have some baselines for comparison. In this section, simple ROC-curves from simple cuts, a Fisher discriminant and a decision tree are found for this purpose. These tests will be done in the simulated data (Monte Carlo) so the actual truth-label can be used to accurately draw ROC-curves. The comparisons will be made using ML1 + ML2 for decisions trees and Fisher while only using the cuts described in section 2.3.1 when cutting. The ROC-curves are calculated using a sample of 10^6 events split in 80-20 train-test sample. The ROC-curve along with calculated AUC-score can be seen in figure 12. .

4.1.2 Reweighting

In this project we used the gradient boosted decision tree reweighter available from the hep_ml Python library [8].

With the right parameters reweighting can provide robust improvements, and render the predictive scores

[20]Here the attribute 'sigma-mp' denotes the proportionality between σ_2 and σ_1: sigma-mp $= \frac{\sigma_2}{\sigma_1}$

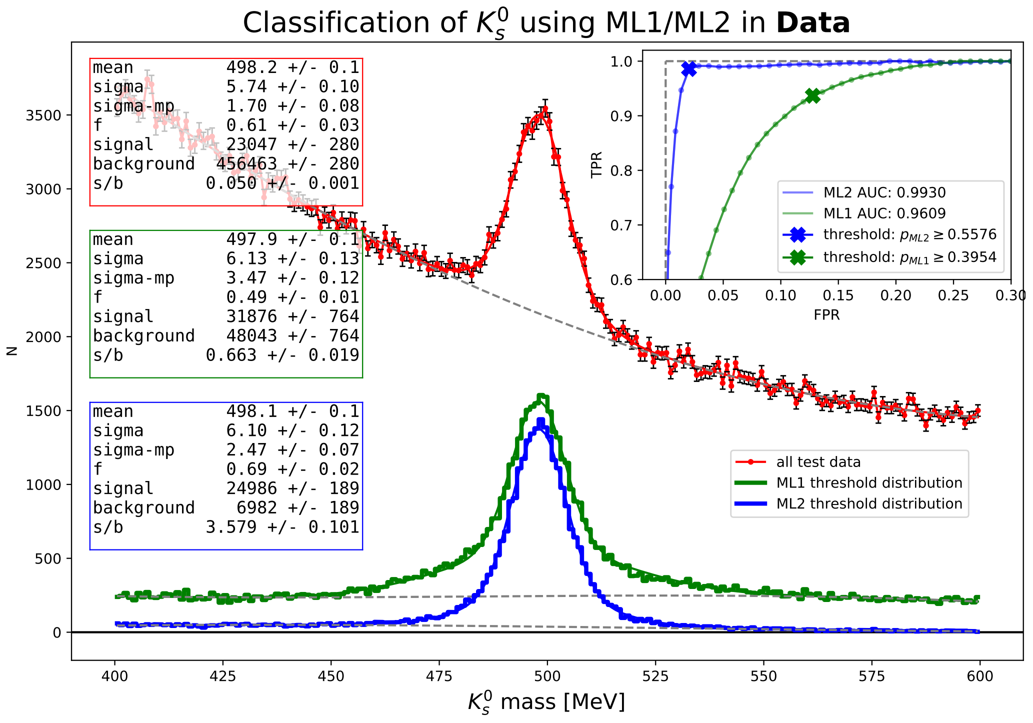


Figure 11: In the main figure, 3 mass distributions along with tables of the fitted values are shown. The red distribution is the collection of all the test data. The green and the blue distributions are subsamples defined by the thresholds shown in the upper right figure. The upper right figure also shows the estimated ROC curves for ML1 and ML2, along with their AUC. Both ML1 and ML2 were trained on different halves of $4.05 \cdot 10^5$ real data events. Description of the attribute 'sigma-mp' can be found in footnote 20.

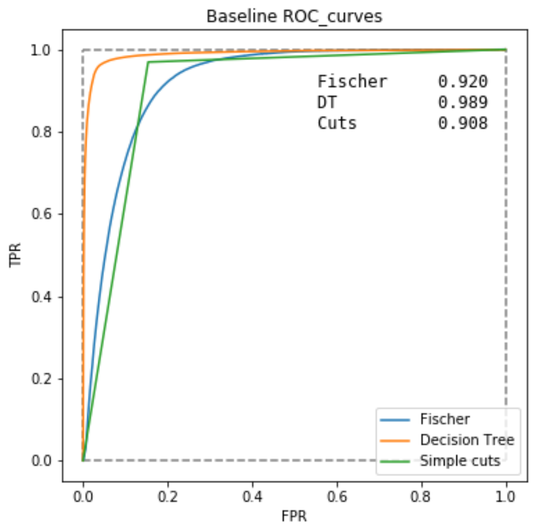


Figure 12: The ROC curves of simple models in **MC**. One done with the cuts described in 2.3.1, one with a decision tree with a max depth of 10 and one with the Fisher discriminant method.

from a model trained in MC more robust. This was especially relevant when applying this to the ML1 set. Here the distributions span the same value ranges and are continuous and thus provide a good basis for a successful reweighting as discussed in section 3.3.1 (see figure 13).

It can also be seen that an XGBoost classifier finds it significantly harder to distinguish between the data/MC ML1 distributions after the reweighting as can be seen in figure 14. The ability for a classifier to predict whether or not an event originates from MC, depending on the variable set and prepocessing method, is summarized in table 5.

Reweighting is not as much of an improvement when applying it to the other variable sets, but it still makes it significantly more difficult for a predictor to tell the difference between real data and MC, as evaluated in table 5

4.1.3 Boosted Decision Trees

We will now apply the gradient boosted forest algorithm, XGBoost. As described in section 3.4 we have

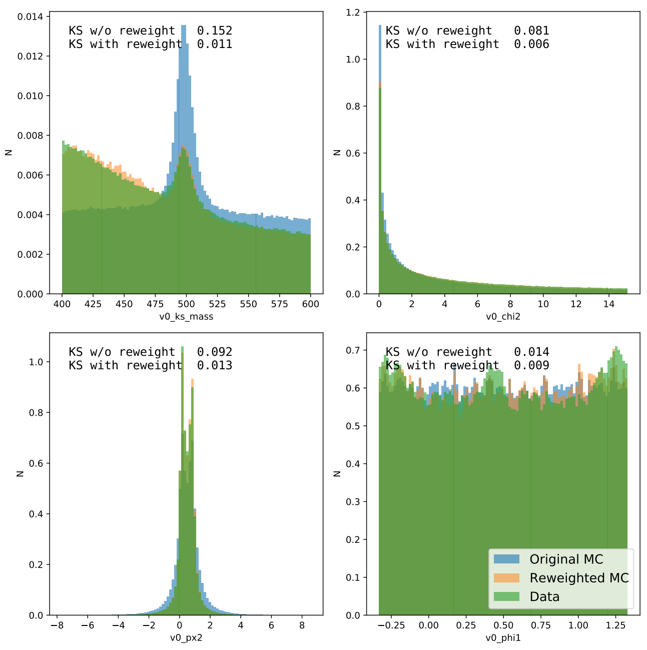


Figure 13: Scaled and reweighted representative distributions for ML1. The reweighting here is very successful and the weighted MC distribtutions are almost identical to the data distribution. The full ML1 sample is available in appendix figure 42

Table 5: First column is the raw distribution. Scaling and reweighting has different effects on our ability to distinguish between data and MC depending on the variable set. The last column is scaled **and** reweighted data

Variable set	AUC$_{raw}$	AUC (scaled)	AUC (+reweight)
ML1	0.91	0.71	0.58
ML2	1	0.89	0.771
ML1+2	1	0.91	0.774

two approaches for training the classifier while still maintaining its applicability to real data. In order to test the classifiers applicability to data we draw ROC-curves based on the method introduced in section 2.4.3.

The first training procedure is to train using MC, and involves training on scaled and reweighted simulated data. Reweighting is introduced in section 3.3.1 and evaluated in 4.1.2. The estimated ROC curves for models trained like this and tested in data can be seen for ML1, ML2 and ML1+2 respectively in figure 15.

One curious finding was that when applying the MC trained model to data, a higher AUC was achieved when preprocessing the test data with the MC feature space transformer instead of the more obvious data transformer. This can only be the case if the important features already lie within the same interval for MC and data, and thus using the same transformer as

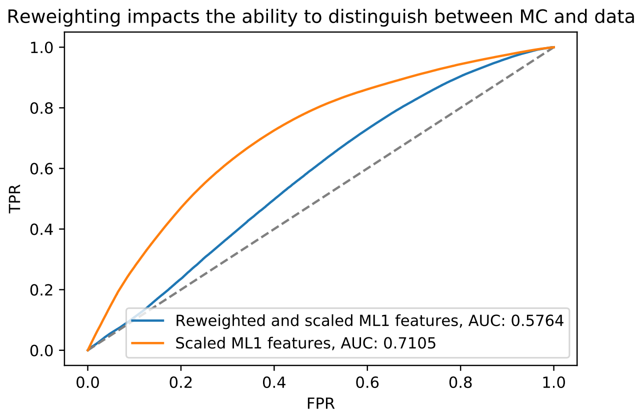


Figure 14: An XGBoost classifier tries to predict the difference between data and MC that is just scaled to match each other, and data and MC that are both scaled and reweighted

Table 6: Scaling and reweighting has little to no effect on the predictive power, except when evaluating the ML1 set. The tests were run on a batch of 10^6 points

Variable set (AUC fit)	AUC (raw)	AUC (scaled)	AUC (+reweight)
ML1	0.889	0.905	0.912
ML2	0.975	0.982	0.982
ML1+2	0.979	0.993	0.994
AUC (cross ML1/ML2)	AUC (raw)	AUC (scaled)	AUC (+reweight)
ML1	0.90	0.93	0.95
ML2	0.91	0.92	0.92

used in training outweighs the minor discrepancies between the training and testing feature spaces.

The second training procedure is to train in data, and involves creating pseudo-labels from the v0_ks_mass feature. The estimated ROC curves in data can be seen for ML1, ML2 and ML1+2 in figure 16 for this training procedure.

We want to highlight that reweighting actually does improve ML1 slightly, and quite robustly across the different evaluation methods laid out in section 3.5. The general changes in predictive power from scaling MC to match data and reweighting can be found in table 6.

4.2 Lambda (Λ) and Lambda-bar ($\bar{\Lambda}$)

Comparing the basic fit of the K-Short with the Lambda particle, clearly reveals that the peak of the Lambda is significantly smaller compared to the background. This gives a good playground for testing the algorithms developed on the K-Short to see if it generalizes when applied to a less significant signal.

Firstly we showcase the results for the method of training on scaled and reweighted MC. The estimated ROC

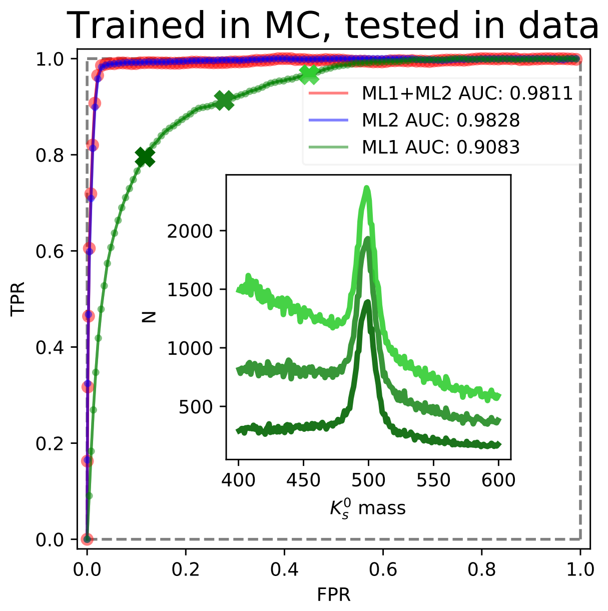


Figure 15: ROC curves for ML1, ML2 and ML1+2 estimated through fitting of the v0_ks_mass feature as described in section 2.4.3. The models were trained in scaled and reweighted MC and applied in data. The mass distribution for three different thresholds (marked with crosses) for the probabilities provided by ML1 are shown in the inserted figure.

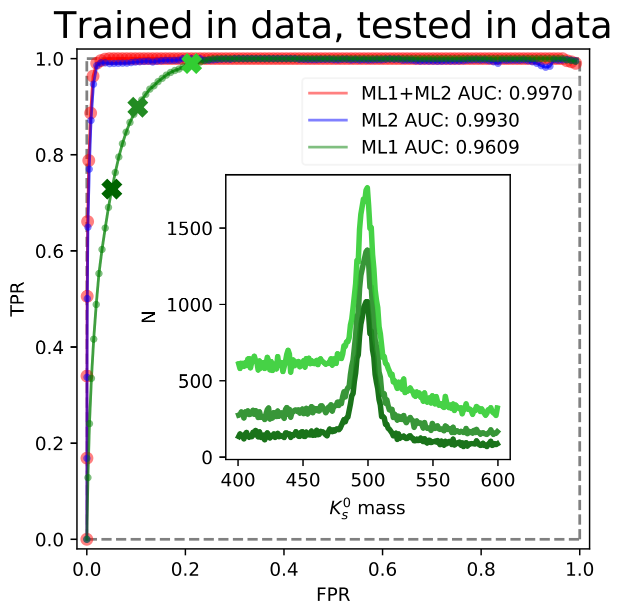


Figure 16: ROC curves for ML1, ML2 and ML1+2 estimated through fitting of the v0_ks_mass feature as described in section 2.4.3. The models were trained in data and applied in data. The mass distribution for three different thresholds (marked with crosses) for the probabilities provided by ML1 are shown in the inserted figure.

curves in data can be seen in figures 17 and 18

The tricky part here was to have our ROC curve estimation method generalize. We found that we needed to limit the fitted signal mean closely to the expected value for the lambda particle, because otherwise, the fit function would try and fit other apparently insignificant peaks, and draw an erroneous ROC curve.

The results for the method of training in data can be seen in figure 19 and 20 where we draw the ROC curves estimated in data.

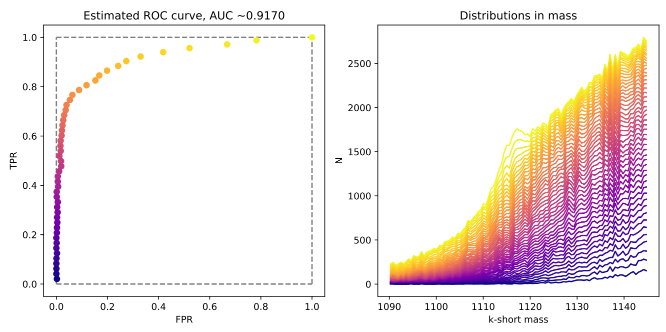


Figure 17: ROC curve estimation from mass of ML1 trained in scaled and reweighted MC for the lambda particle.

4.2.1 Model Performance on the Lambda

After training an XGBoost-classifier on the least correlating parameters in the simulated data, we got the ROC-curve and AUC score seen in figure 39 using truth-labels. To obtain a smooth probability distribution (see figure 18) it was necessary to exclude cosTheta, pv0_x/y/z and ntrk_pv0 from the reweighting procedure (see section 3.3.1 for more insight into this).

In addition, it became clear that the Lambda-particles relates to some parameters that the K-Short does not. The full feature-importance tables can be seen in the appendix in figure 30.

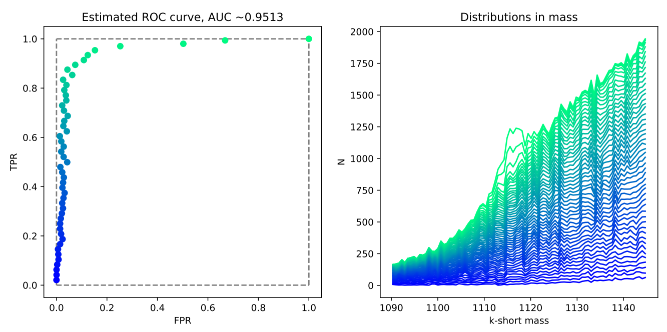


Figure 20: ROC curve estimation from mass of ML2 trained in data for the lambda particle.

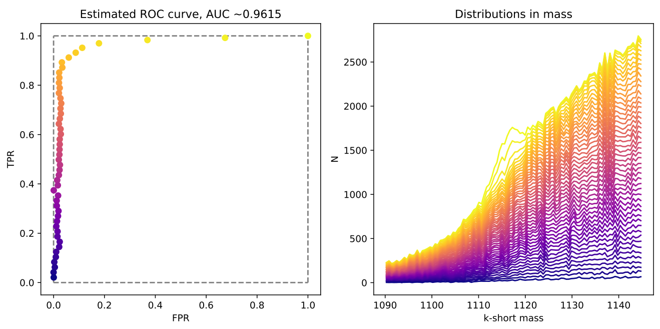


Figure 18: ROC curve estimation from mass of ML2 trained in scaled and reweighted MC for the lambda particle.

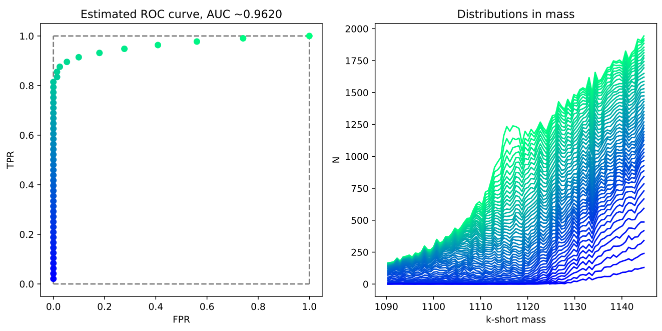


Figure 19: ROC curve estimation from mass of ML1 trained in data for the lambda particle.

4.3 Cross-validating Models

To evaluate models in data we need to build up tools that are robust and verifiable. In this section we will evaluate how the model predictions can be used together and their respective correlations.

4.3.1 ROC - validation

Employing the methods described in section 2.4.3, we get the following (see figure 21):

1. ML1/ML2 cross-validation where one model is used to check the performance of the other. Both are trained and used in data.

2. Training a model in MC on ML1+2, we can use it to validate the predictions of a model trained in data on ML1+2 variables. This mainly validates whether or not our models agree, more than their predictive power, but in conjunction with other validation methods, this still gives valuable insight in both our ability to use MC-trained models in data, and whether or not we extract most of the extractable information in the variable set.

Even though boosted decision trees show resilience towards erroneous labeling when the mis-labeling **only** occurs within one of the two categories[21], it could possibly still be advantageous to reduce the ratio of mis-labeling. To do this, two set of parameters with minimal inter-pairwise correlation is selected. Then a model with the first set of parameters is trained on pseudo-labels and cuts are made in the prediction scores of new data to select data points which it is fairly confident is signal and likewise background. The second model, with the second set of parameters, is then tested on a third dataset after being trained on the second dataset with labels generated by the first model. The signal mis-labelling ratio went from < 1 (~ 0.58) to $\gg 1$ (~ 4.28), but whether the second model was trained on the first ratio or the second ratio had no impact, the accuracy remained the same.

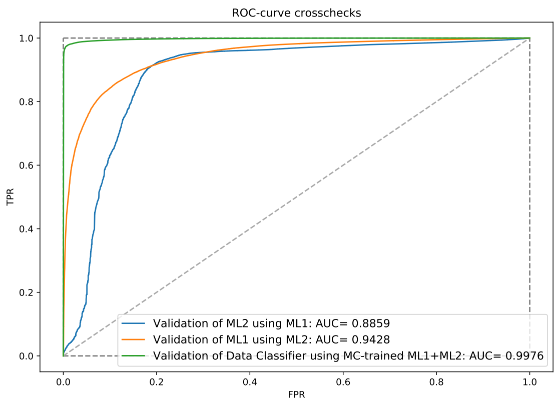


Figure 21: ROC-curves for cross-validating model performance. As described in section 2.4.3, the worse classifier usually ends up giving a worse ROC-curve, even if the model it is validating is better. The ROC-curve for cross-validating model performance between MC and Data, using MC as validation shows that the classifiers agree almost completely, giving good confidence in our model's ability to extract most of the relevant information from the real data.

Note that we cannot exclude that this would still be impactful in situations where the signal mislabelling ratio becomes arbitrarily large, but from our analysis it seems that it does not have a great impact when the other class is correctly labelled.

4.3.2 Correlations between model predictions

Yet another way of investigating the models and their interactions is to investigate the correlations between what points the different models predict as background and signal, and with what confidence. Ideally, since

[21]With the procedure used for generating pseudo-labels, the signal category is the one that contains a lot of background as well.

for the K_S^0-particles, the ML1 and ML2 variable sets are uncorrelated, their exact scores should not be correlated for either signal or background. We test this by calculating the MIC-score for predictions from XGBoost models trained on ML1, ML2 and ML1+2. We do this for K_S^0 where we train and test in data, K_S^0 where we train in reweighted MC and test in data, and Λ where we train and test in data. ML1 and ML2 were trained on different sets of points of equal size, and ML1+2 were trained on both. All were tested on the same points. The results for K_S^0 data to data can be seen in figure 22. The other figures can be found in appendix figure 43 and 44 .

As can be seen in the figure, ML2 and ML1+2 correlate heavily in their signal prediction, which is to be expected since the final prediction of ML1+2 is dominated by ML2. We see that the predictions of ML1 and ML2 are **not** very correlated, as desired. Background and signal is taken as points that both models classify as belonging to either class.

4.4 Final Mass Estimate

Now, having refined and optimized our classification methods, we feel ready for a final estimate of the particle-masses. For this calculation we will use two XGBoost models purely trained in data to provide us with K-Short candidates. The two XGBoost models are trained separately on the ML1 and ML2 feature spaces with different training data. We then select the 100 most certain K-Short candidates in the test data, where the metric of certainty is
$\sqrt{(p_{ML1}^2 + p_{ML2}^2)/2}$ with $p_{ML1/ML2}$ the prediction from ML1 and ML2 respectively. The 100 most likely candidates were selected out of an original 479510. These candidates then have a corresponding v0_ks_mass and v0_ks_massErr from which a weighted mean and a corresponding χ^2 can be calculated, as is done in figure 23.

We used the 'massErr' parameter, as our previous calculations (figure 5) has shown that the CERN-estimated errors are very closely related to the actual uncertainties. It should be noted that the values of the errors of the candidates lie within the interval that was investigated.

The same procedure is applied to the lambda particles in figure 24. However, the candidates included 7 clear outliers as indicated by the cross-marker, which were not included in the weighted mean. The 93 candidates were selected from an original 85680.

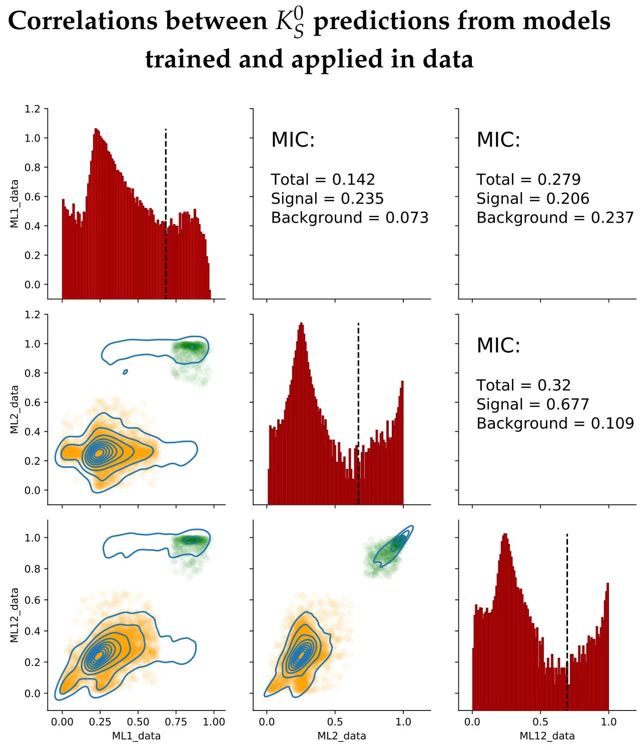


Figure 22: MIC-correlation between different model outputs. ML1 and ML2 were trained on different sets of points of equal size, and ML1+2 were trained on both. All were tested on the same points. On the diagonal, probability density distributions with log(N) on the y-axis is drawn with a line to indicate the background/signal split. In the lower half, contour plots for all prediction scores are plotted as well as classified background (orange) and signal (green). Background and signal is taken as points that both models classify as belonging to either class. In the upper half, summary MIC-scores for all scores, background only and signal only are shown

This gives the final mass-estimates as:

K_S^0: $498.1 \pm 0.5 MeV$
$\Lambda/\bar{\Lambda}$: $1116.15 \pm 0.18 MeV$

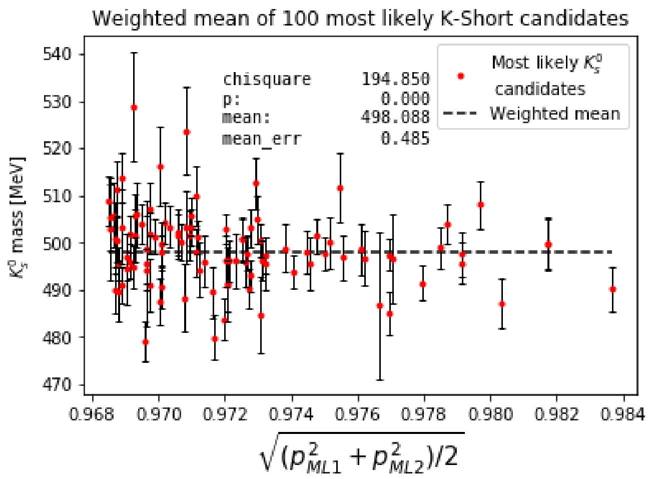


Figure 23: Weighted mean of the 100 most likely K_S^0 candidates provided by ML1 and ML2, both trained in data but on different sets. The metric of likelihood is $\sqrt{(p_{ML1}^2+p_{ML2}^2)/2}$ with $p_{ML1/ML2}$ the prediction from ML1 and ML2 respectively. The y-axis is the v0_ks_mass for the points and the errorbars are the associated v0_ks_massErr

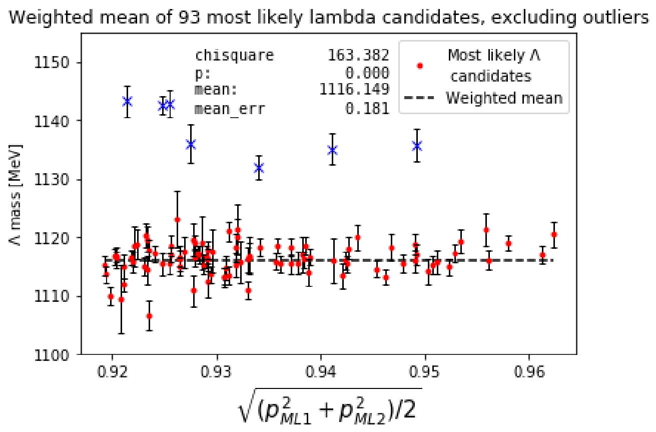


Figure 24: Weighted mean of the 93 most likely Λ candidates provided by ML1 and ML2, both trained in data but on different sets. The metric of likelihood is $\sqrt{(p_{ML1}^2+p_{ML2}^2)/2}$ with $p_{ML1/ML2}$ the prediction from ML1 and ML2 respectively. 7 outliers were discarded for the calculation of the weighted mean. The y-axis is the v0_la_mass for the points and the errorbars are the associated v0_la_massErr

5 Discussion

In this section, the results will be discussed, in order to evaluate the success of the models and which changes would have been beneficial looking at the project in retrospect. The discussion will end with suggestions for future work.

5.1 Correlations

The analysis of correlations in the data yielded very insightful information, especially in this case where the data set is undocumented. Placing the groups in categories gave the possibility of determining if a feature should be discarded based on the group it belonged to[22].

5.1.1 Correlations with Mass

In this project the method for labeling the training sample in the data was based upon the mass. For this reason it was necessary to decorrelate the feature space with the mass to not end in a situation like figure 6 where the mass and not the signal itself, is fitted. Even though the performance proved very strong with this type of label, throwing out features correlated with mass have no doubt weakened the models significantly. But in a data set where no truth label is provided this proved to be necessary.

5.1.2 Correlations Between Predictions

As can be seen in figure 22, we successfully decorrelated the prediction scores for signal and background between ML1 and ML2 for the classification of K_S^0-particles, as was described in section 4.3.2 as we had hoped for. It is expected that there is some correlation since both classifiers are useful, but the fact that this correlation is solidly below 0.5 is encouraging[23].
We also see that this same figure indicates that there is a significant aspect of disagreement between what ML1 and ML2 classifies as signal (which can be seen as the long horizontal signal contour). This is especially due to the fact that ML1 in general classifies less points as signal, and more as background, because it generally has less predictive power.

[22]A good example of this is the 'a0', 'a0z', 'a0xy' set where most information can be contained in one

[23]For two perfect classifiers the correlation would be 1, since they would give 1 for all true signal points and 0 for all true background points

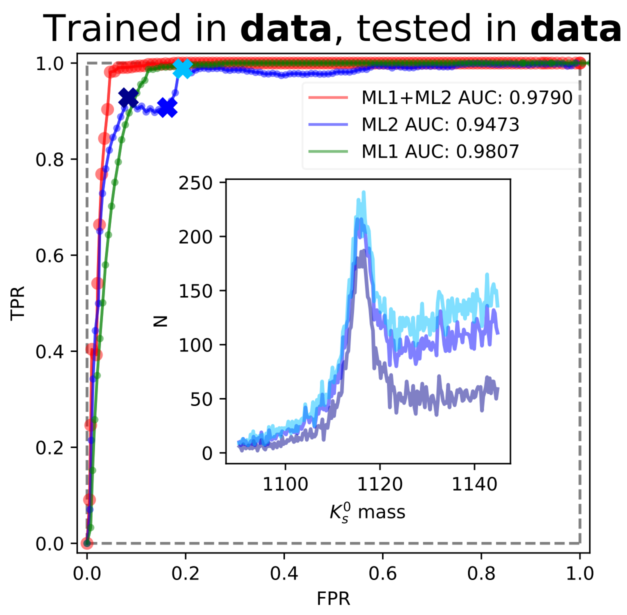


Figure 25: Classifying Λ particles by training XGBoost models in data, with the features of ML1, ML2 and ML1+ML2. Here we see that the features of ML2 correlate with v0_la_mass.

As would be expected from other evaluation methods, we see that the correlation for signal classification between ML2 and ML1+2 is very high (a MIC-score above 0.7 is high [7])

5.1.3 Lambda Features

When using our tools on the Lambda and Lambda-bar, we erroneously reused the ML1 and ML2 features for training, believing the internal correlations to be somewhat comparable to the K-Short. As we calculated the MIC-scores and SHAP-values, it quickly became clear that this was not the case. 'v0_pt2' and 'pT' had heightened impact on the training, while 'Alpha/alpha' and 'thetastar' were significantly more correlated with mass. If we had allocated more time to the classifications of Lambda-particles, running a proper analysis would have been feasible. The correlations can be seen in table 8 and the most important features can be seen in figure 30.

Our predictions in this case were still strong and did not lead to serious misclassification of background as in fig 6, but we did see some correlation with mass. More predictions were in general less confident, which we attribute to the smaller amount of signal. The prediction landscape and correlations changed significantly (see figure 44). Here, using ML1 to validate ML2 would be inappropriate since they are heavily correlated.

5.2 Model Evaluation Methods

The issue of no confident true labels existing in data proved a challenge on both sides of the actual classification. After a model was tested, it was not clear how to accurately quantify the quality of a classification. In this project two methods for making ROC-curves without labels were introduced.

Firstly, the signal and background could be estimated using a fit of the peak with a double Gaussian fit on top of a polynomial background. This proved to be reliable as seen in figure 10 where the estimated AUC was around 0.01 off. Thus this method was the main indicator for the quality in the other results. This method should also be easily applicable to other peaks as seen when the method was modified to work with the Lambda peaks in section 4.2.

Secondly, the use of cross-validating two uncorrelated models was tested. Doing this with ML1/ML2 gave a reasonable estimate on the least accurate of the models, whereas the result for the better model seems uneven. This is a result of the more accurate model classifying the clear signal and background correct and then classifying signal with different amount of random labels in the middle. Even though this method provides some insights, the first of the tested methods proved more robust and accurate. A flaw with the second model is also that the feature spaces should be split in two completely uncorrelated models, whereas the first can take the entire feature space as long as it is not correlated with the mass.

5.3 Advantages of the Monte Carlo Simulation

The MC data set gave us an opportunity to obtain a strong intuition for the signal and background distributions, as well as providing an independent validation method for much of our work.

Going through the trouble to make MC useful for classifying things in real data was very rewarding and even though it turned out that many of the variables that benefitted strongly ('pv0_x', 'pv0_y','pv0_z', 'ntrk_pv0') were not very useful for classifying events, since they are very noisy parameters as can be seen in figure 41a.

Reweighting can be quite a lot of work and requires careful consideration, and often renders negligible improvements but nonetheless, it rarely worsens the classification and might improve it.

5.4 Accuracy

5.4.1 Accuracy of Mass Predictions

We can compare the fits from section 4.4 to the values given in a review paper ([9]):

$$K_S : 497.611 \pm 0.013 MeV$$
$$\Lambda / \bar{\Lambda} : 1115.683 \pm 0.006 MeV$$

Comparing this to our fits, we have $z = 0.98$ and $z = 2.59$ respectively ($z = \frac{|\mu_1 - \mu_0|}{\sqrt{\sigma_1^2 + \sigma_2^2}}$), which are both not completely unreasonable.

5.4.2 Choice of Model

In section 2.3.1 we discussed the cuts based on physical intuition found in the ATLAS paper. We unfortunately heard of these late, but having made these cuts from the very start could potentially have improved the models. However, this project was mainly concerned with attacking the problem from a "blind" approach, using quantitative principles whenever possible. This of course somewhat worsens our performance, but nonetheless means that we are not biased towards any prior physical understanding that would hinder new discoveries.

In most cases we have refrained from using the simple cuts before using any classifier, since the classifier is more robust if given a lot of high-confidence background.

In general we used XGBoost due to its general higher performance (see table 7) and relatively fast training time.

5.5 Future Work

In this project many methods were tried, but the realm of machine learning is rapidly expanding and new methods are introduced often. The project could easily have continued into the field of Neural Networks which come in many forms. This was tried shortly (see sec 7.1.6 in appendix), but it became clear that much more work in the feature space had to be done before any neural network could compete with the Boosted Decision Trees.

The Boosted Decision Trees also have many hyper parameters which were only understood but not optimized in this project, except for little experiments (see appendix section 7.1.5). Much improvement could potentially be found here, but besides letting the number of estimators approach infinity, we had no short, clear answer for an improvement that wouldn't lead to overfitting.

This project was done with only a superficial overview of the field of High Energy Physics. There is definitely a lot to gain here. As described in section 2.3.1 a lot of cuts are already made in the data, but many other could be beneficial but was not done in this project. Looking back at the project, it becomes clear that features like pseudorapidity which was just discarded, could have been used to make a cut in the data to improve the quality of remaining data. The feature space also included features such as number of hits in the different layers, which further could have reduced the number of events with poor quality.

Most of the methods in project was developed and tested on the K-Short particle, but tried on the Lambda. However, a lot of further analysis could have been made of the Lambda particle. In addition, the lambda-bar particle was barely touched in the project, even though comparison of the results from Lambda and Lambda-bar particle would have been of great interest.

One way of improving our prediction of the mass of the Lambda/Lambda-bar is using the fact that their masses must be equal. Thus we could have combined two *independent* final mass estimates with uncertainties into one total estimate of the Lambda invariant mass. This also provides a reliable validation of our mass estimates for Lambda and Lambda-bar.

Furthermore, we would like to dig more into the track reconstruction itself.

6 Conclusion

Throughout this project the main finding is not necessarily a single result but the use and experience with the different methods of attack on a very general problem: Determining signal from noise in a data set with a large feature space.

In the real data a lot of experience was gained exploring the robustness of Boosted Decision Trees to "pseudo-labels" based on mass. This required the investigation of the features to make sure they were not correlated with this result. Without labels in the Data it was also a challenge to evaluate the accuracy of a model. This was primarily done by using fits to determine the signal and background in order to estimate a ROC-curve. This estimate proved very valuable and was close to the actual value when tested in Monte Carlo.

The generalisability of the ideas of the project especially showed when applying all of the tools we had developed for K-Short peak finding for Lambda/Lambdabar peak-finding, which was done primarily in the last week of the project without great issues.

One would assume that using the full parameter space couldn't possibly lead to a worse model for predicting peaks, but during the course of the project it has turned out that correlations between the mass and the parameters used for classification generally should result in those parameters being taken out of the model, since this leads to high disturbances in the central investigation of the mass peaks. This is especially important when using mass as a label, but generalizes to MC-models that are to be applied in data, as well as any model using any specific parameter to create pseudo-label.

Having a large amount of data helps every aspect of statistical problem solving. Especially a great number of well-simulated data points makes life easier, as the truth label enables testing different classification-methods in a quicker and clearer way.

No matter how well a Monte Carlo simulation works, there will always be some discrepancy between the created data and real life. We have found that this problem should not be seen as a barrier, as it can be negated by creating surprisingly certain truth labels solely in data.

If one were to cross-validate a non-perfect classifier with another non-perfect classifier, the resulting ROC-curve is likely to be a product of the accuracy of the model used for validation, more than the classifier one wishes to evaluate.

Another general takeaway is that it pays off to become comfortable with the data before applying the larger statistical machinery. While the amount of time invested in correlations rendered great improvements in our result, it would probably also have improved the result if notions of data quality, such as a low pseudorapidity, were considered.

6.1 Acknowledgements and Final Remarks

First of all, we would like to thank our advisor, Professor Troels C. Petersen, for inspiring us to take up this very interesting problem through the Applied Statistics course, and for continued support and inspiration throughout the project.

Secondly we would like to thank each other for many weeks of hard work, dedication and clever ideas. It has been fun, challenging and very rewarding to try to come up with, and validate, all of our ideas.

We would also like to express our respect for the ATLAS team. Their results are incredible, and we have but spotted the tip of the iceberg of the amount of work that goes into creating, and cross-validating their results.

7 Appendix

7.1 Honorable Mentions

During the course of the project, several ideas were scrapped for either being unfruitful or too large to tackle. Many of the things we spent significant amounts of time on never made it into the report or were only mentioned as passing remarks. Most had some interesting lessons learned anyways, so this section is dedicated to the discarded ideas with the most potential.

7.1.1 XGBoost on UMAP

After playing around with UMAP, a dimension-reduction algorithm, we realised a simple, visual cut in a two-dimensional representation was surprisingly good at separating signal from background. We set out to try training a boosted decision tree on transformed data in a variety of different dimensions. The main motivation behind the idea, was the fact that a supervised UMAP can work towards maximizing separation in parameter space, thereby improving the value of every leaf in the boosted forest.

As we only had eight weeks, and this was not the purpose of the project, we had to stop work on this.

7.1.2 uBoost

About six weeks into our eight week project we found a paper titled ”uBoost: A boosting method for producing uniform selection efficiencies from multivariate classifiers”[10]. This paper seemed to solve the problem of decorrelating a BDT with the mass without removing information in the form of correlating parameters. While the algorithm is immensely slow, the advantages can clearly be seen in figure 26. uBoost builds upon the AdaBoost-framework so the AdaBoost algorithm was used when comparing.

7.1.3 Uncertainty in Data

In figure 5 we found a constant offset between the uncertainty we find in the data and the uncertainty estimated by CERN. This discrepancy was never accounted for.

7.1.4 COVID Modeling

During the recent spike in COVID-19 infections in Denmark, the group contributed about a week's worth of time to fitting and improving a SEIR-model for different local outbreaks in order to evaluate the effects of different lockdown measures. This was complicated by the fact that most numbers are not confidently tracked and so the data is very noisy. Furthermore, the uncertainties were in general not known and this presents a large issue. During this week we mainly gained insight into speeding up differential equation solvers, and how to produce confidence intervals on a parameter that we could not fit by using a Monte Carlo simulation on the parameters that we could fit. This could then be used to estimate whether or not the uncertainty on the number of positive tests was estimated correctly. This issue of providing uncertainty estimates for parameters where they simply do not exist is sure to show up again, as it does all the time in e.g. astrophysics.

7.1.5 Hyper-parameter Optimization

Most of the work with the Boosted Decision Trees was done using out-of-the-box implementations from various sources. Some parameters were tweaked, but the algorithms have hundreds of variables to control and many interesting features. Due to the interconnectedness of the parameters, searching trough hyper-parameter space is

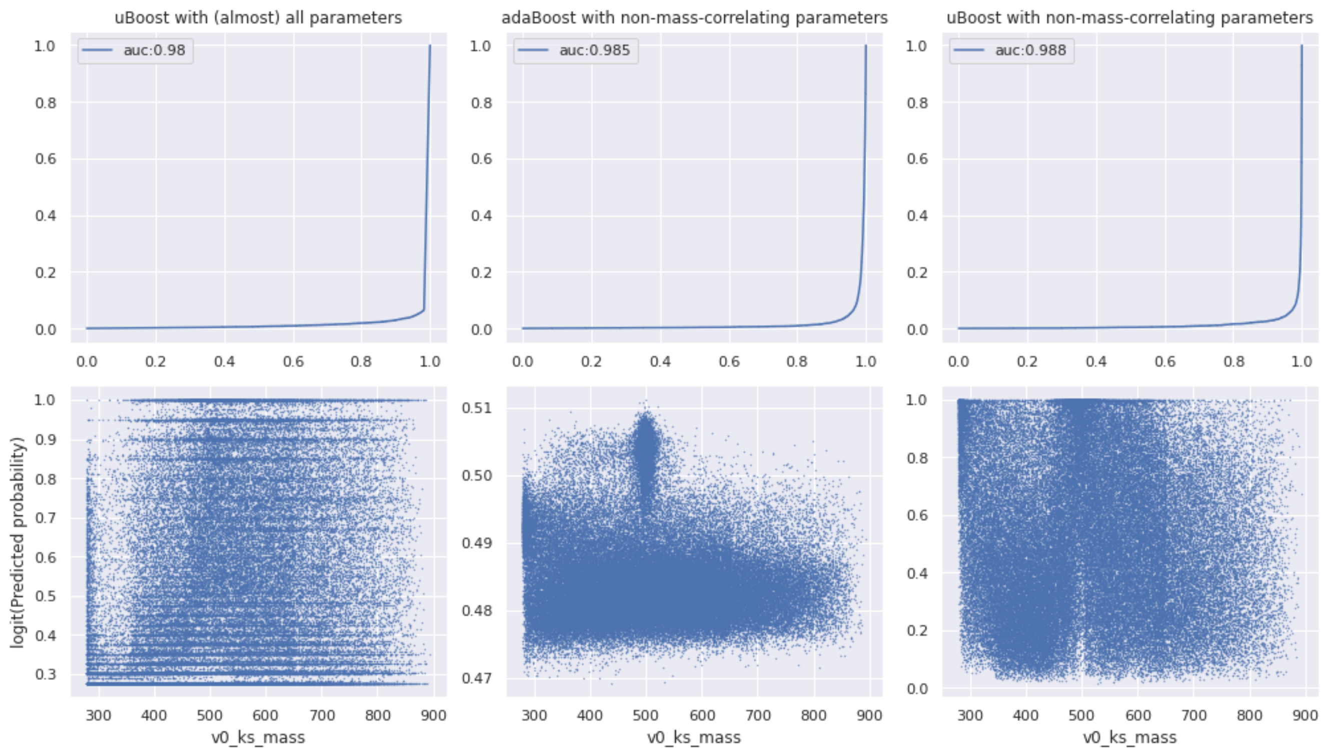


Figure 26: It is clear from the bottom row of plots, that the trained uBoost are less correlated with mass than even the least correlated group of features (here ML1+ML2) without sacrificing any prediction capability.

very computer-intensive compared to the performance gain. We have mainly resorted to the default parameters, sometimes adjusting settings based the advice from this brilliant internet resource[11].

7.1.6 Neural Networks

Following the trends of the times, we wanted to see if neural networks could improve the classification of the Boosted Decision Trees. After some work using Keras and PyTorch, we quickly noticed this would be beyond the scope of the paper, as we found training times to be orders of magnitude larger without any obvious improvements.

7.1.7 PCA before XGBoost

Much in the vein of 'UMAP XGBoost', running the analytical PCA could improve the training of decision trees. We tried but it gave no obvious improvements.

7.1.8 Comparing the algorithms

We found that comparing the different Boosted Decision Trees could tell us something about what to use during the course of this project. The results were surprising as there was quite a stark difference between the algorithms (see table 7). We did not know what to do with the information, so this ended up with the honorable mentions.

	t_{train} [s]	$t_{predict}$ [s]	auc
AdaBoost	371	8.58	0.982
XGBoost	108	1.12	0.993
LightGBM	6.57	0.467	0.990

Table 7: Comparisons of the three main BDT-algorithms. Trained on $2 \cdot 10^6$ data points with the features from ML1+2. The prediction was for 10^6 MC data points and the AUC is calculated using truth-labels.

7.2 Appendix (Figures and tables)

7.2.1 Distributions of Data and MC

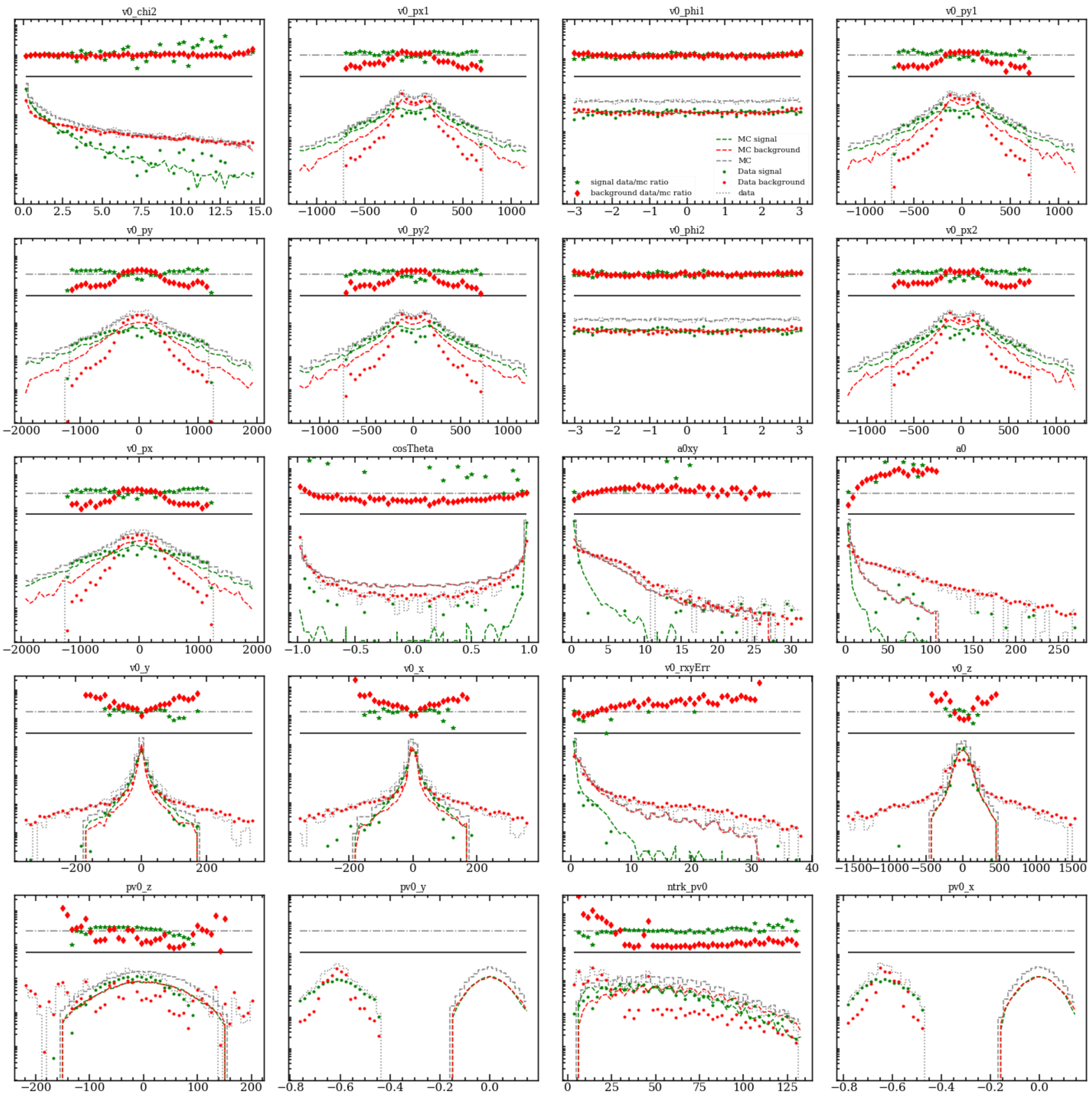


Figure 27: Comparison of distributions of signal and background in MC and data for given features. The top part shows the ratio between the signal (green) and background (red) distribution between data and MC

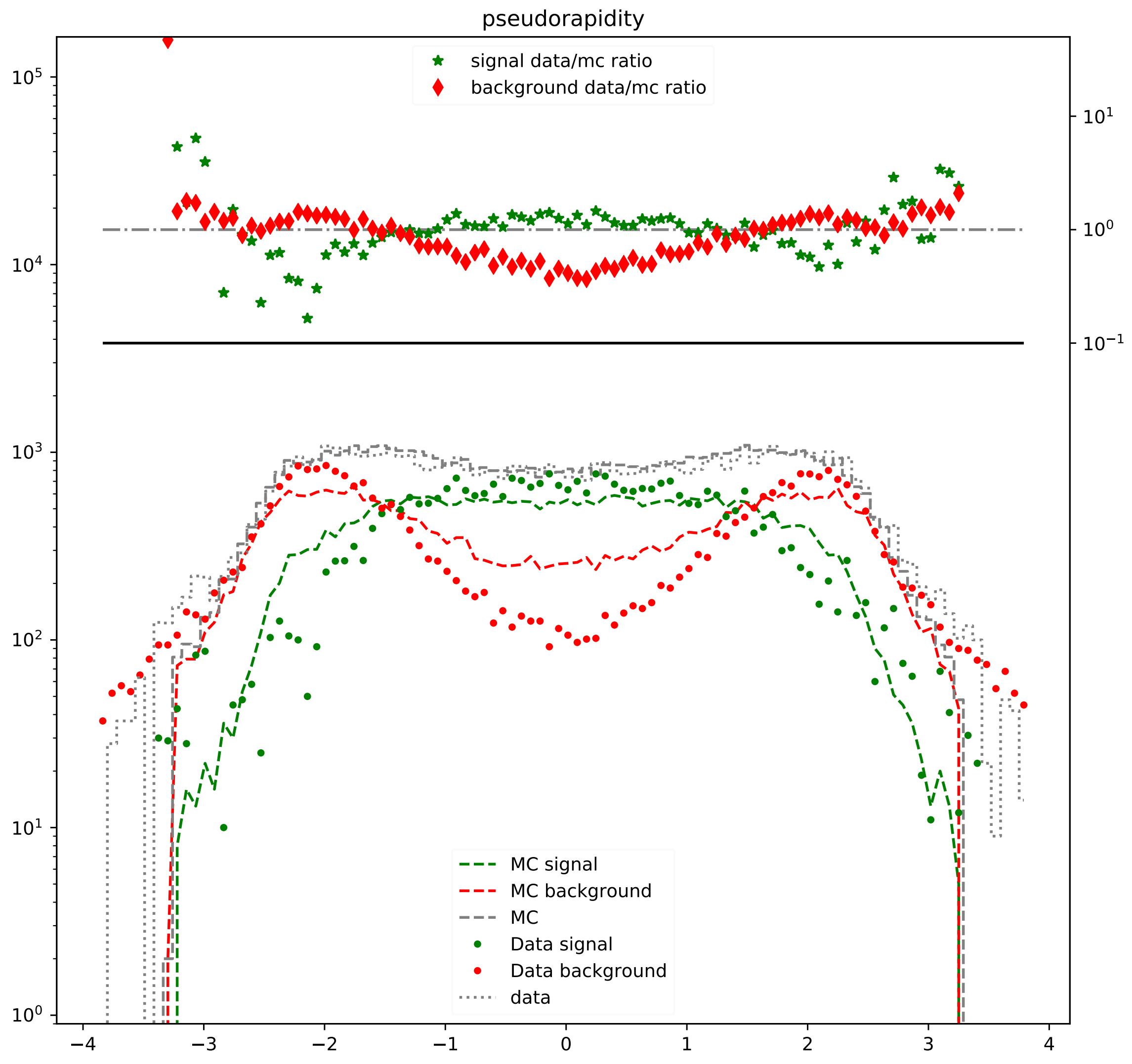
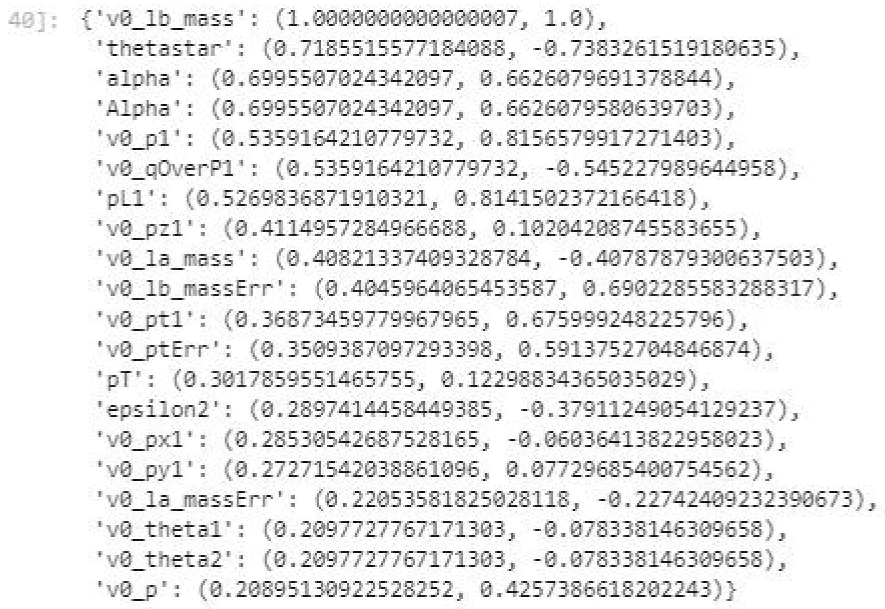


Figure 28: How well the signal distribution estimation for data works compared to MC, given that the underlying distributions for MC and data are equal.

7.2.2 Correlation tables for Λ and $\bar{\Lambda}$

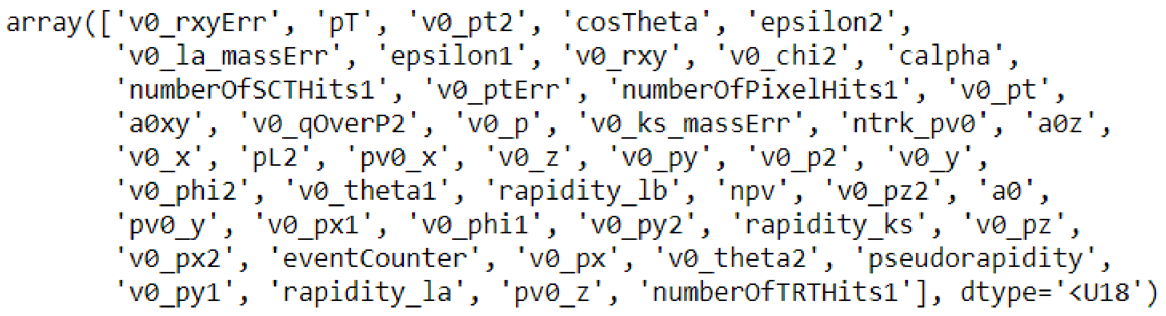
Table 8: Table displaying the correlation with v0_la_mass in MC using both the MIC method as well as the linear Pearson correlation

Feature	MIC (Data)	ρ (Data)
thetastar	0.71	0.74
alpha/Alpha	0.71	-0.66
v0_p2	0.57	0.81
v0_qOverP2	0.57	0.60
pL2	0.55	0.81
v0_pz2	0.44	0.12
v0_la_massErr	0.43	0.67
v0_lb_mass	0.41	-0.41
v0_pt2	0.38	0.66



```
40]: {'v0_lb_mass': (1.0000000000000007, 1.0),
      'thetastar': (0.7185515577184088, -0.7383261519180635),
      'alpha': (0.6995507024342097, 0.6626079691378844),
      'Alpha': (0.6995507024342097, 0.6626079580639703),
      'v0_p1': (0.5359164210779732, 0.8156579917271403),
      'v0_qOverP1': (0.5359164210779732, -0.545227989644958),
      'pL1': (0.5269836871910321, 0.8141502372166418),
      'v0_pz1': (0.4114957284966688, 0.10204208745583655),
      'v0_la_mass': (0.40821337409328784, -0.40787879300637503),
      'v0_lb_massErr': (0.4045964065453587, 0.6902285583288317),
      'v0_pt1': (0.36873459779967965, 0.675999248225796),
      'v0_ptErr': (0.3509387097293398, 0.5913752704846874),
      'pT': (0.3017859551465755, 0.12298834365035029),
      'epsilon2': (0.2897414458449385, -0.37911249054129237),
      'v0_px1': (0.28530542687528165, -0.06036413822958023),
      'v0_py1': (0.27271542038861096, 0.07729685400754562),
      'v0_la_massErr': (0.22053581825028118, -0.22742409232390673),
      'v0_theta1': (0.209772776717l303, -0.078338146309658),
      'v0_theta2': (0.2097727767171303, -0.078338146309658),
      'v0_p': (0.20895130922528252, 0.4257386618202243)}
```

Figure 29: I should be a part of table 8



```
array(['v0_rxyErr', 'pT', 'v0_pt2', 'cosTheta', 'epsilon2',
       'v0_la_massErr', 'epsilon1', 'v0_rxy', 'v0_chi2', 'calpha',
       'numberOfSCTHits1', 'v0_ptErr', 'numberOfPixelHits1', 'v0_pt',
       'a0xy', 'v0_qOverP2', 'v0_p', 'v0_ks_massErr', 'ntrk_pv0', 'a0z',
       'v0_x', 'pL2', 'pv0_x', 'v0_z', 'v0_py', 'v0_p2', 'v0_y',
       'v0_phi2', 'v0_theta1', 'rapidity_lb', 'npv', 'v0_pz2', 'a0',
       'pv0_y', 'v0_px1', 'v0_phi1', 'v0_py2', 'rapidity_ks', 'v0_pz',
       'v0_px2', 'eventCounter', 'v0_px', 'v0_theta2', 'pseudorapidity',
       'v0_py1', 'rapidity_la', 'pv0_z', 'numberOfTRTHits1'], dtype='<U18')
```

Figure 30: Most Valuable Parameters in descending order

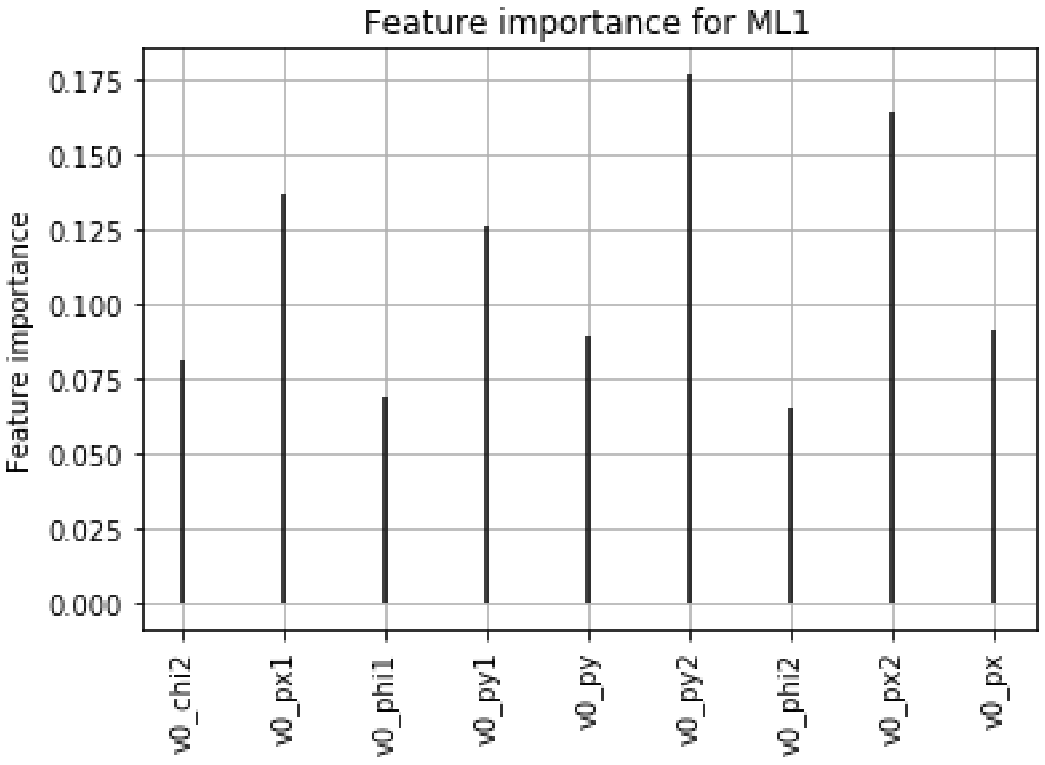


Figure 31: Feature importance for ML1 trained in data for lambda

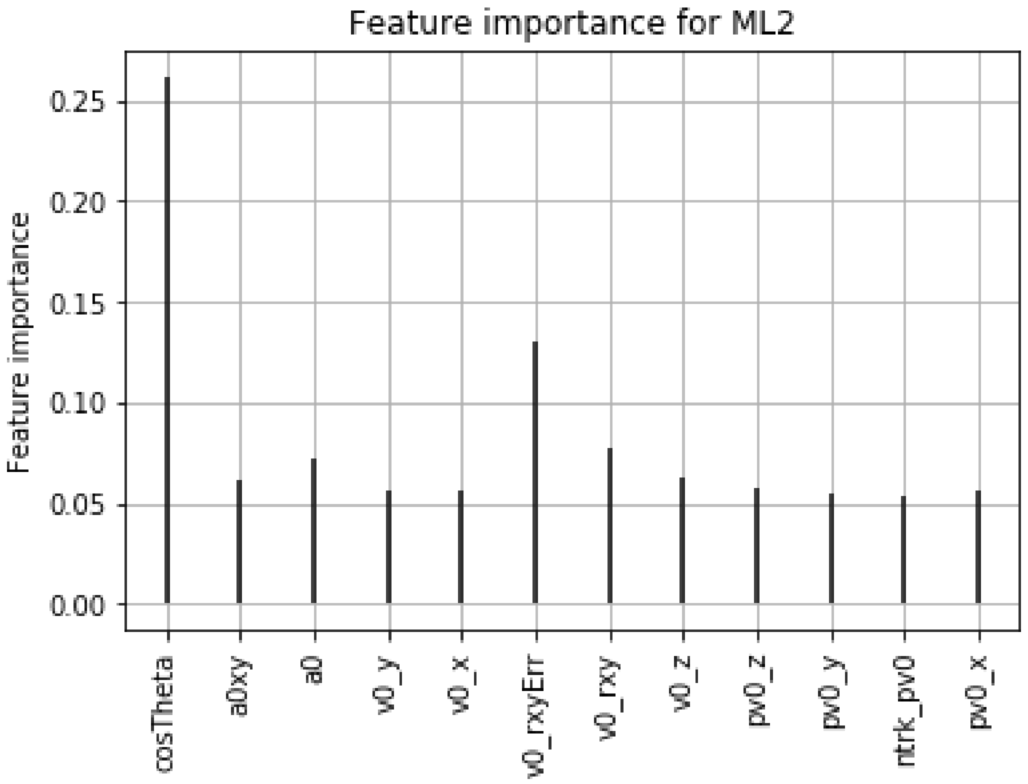


Figure 32: Feature importance for ML2 trained in data for lambda

7.2.3 Signal Distribution

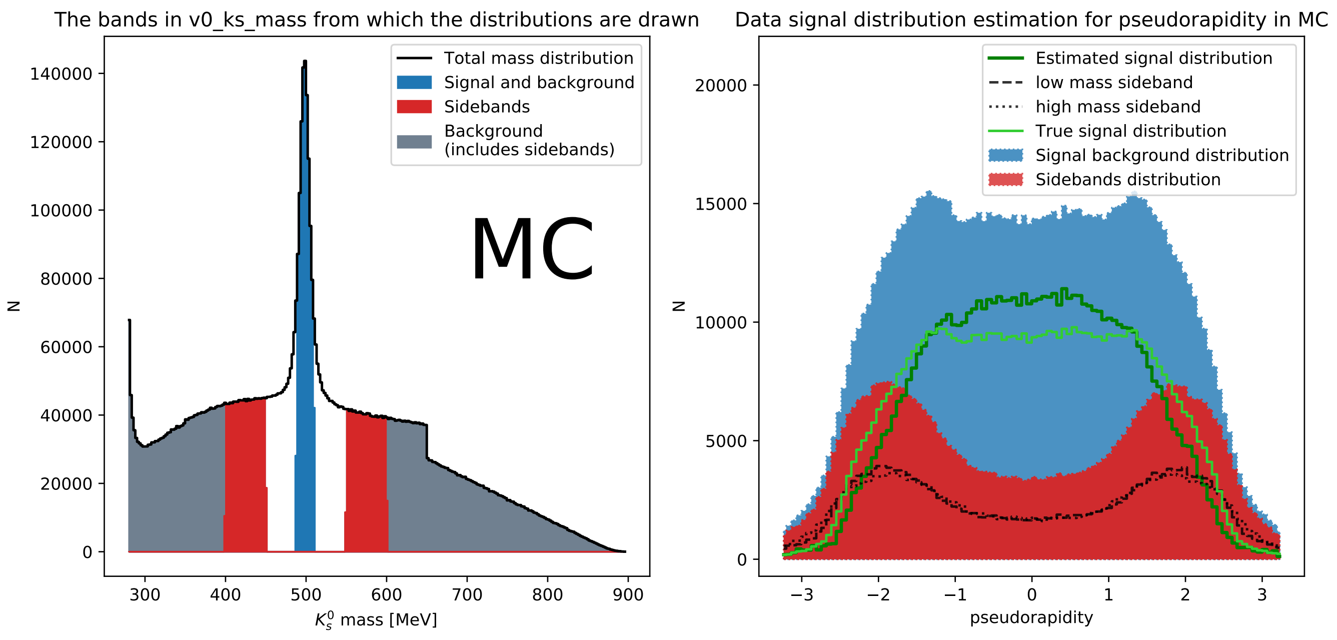


Figure 33: Here we show the same procedure as in figure 9 but for MC instead of data, in order to investigate the validity of the method. The added limegreen line in the right plot is the true distribution for MC, also for the feature pseudorapidity.

7.2.4 Model comparison

7.2.5 Reweighting

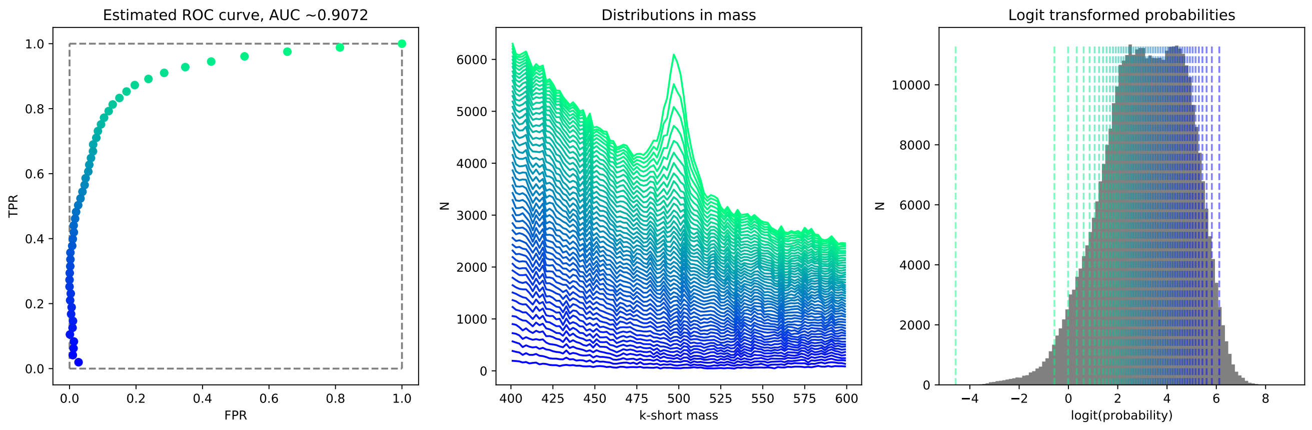


Figure 34: ROC curve estimation from mass of ML1 trained in scaled MC for the K-short particle.

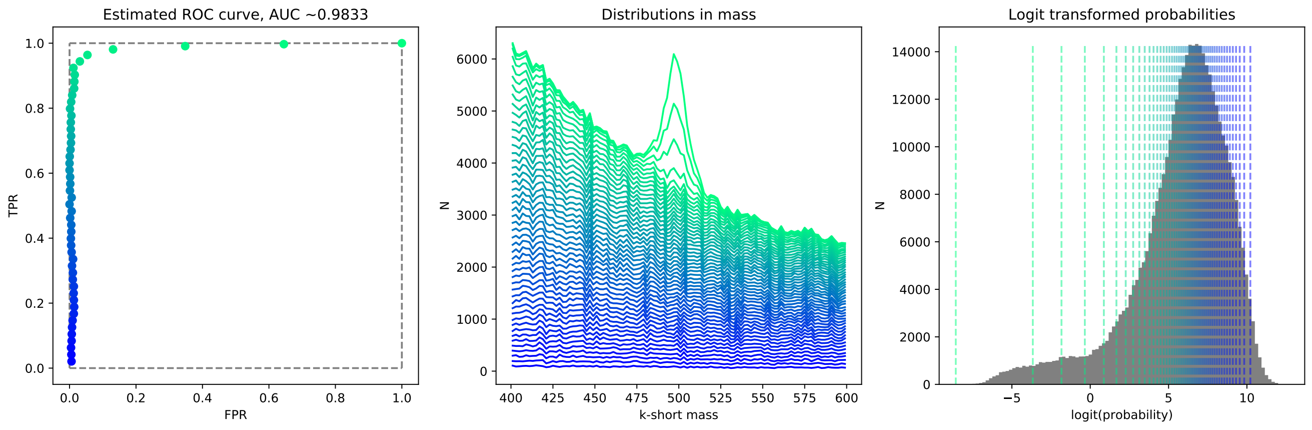


Figure 35: ROC curve estimation from mass of ML2 trained in scaled MC for the K-short particle.

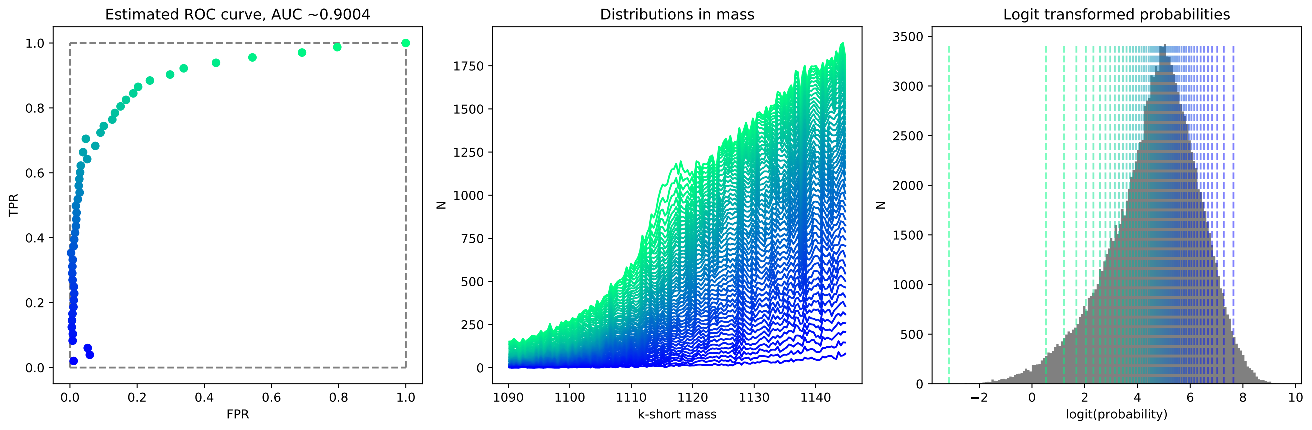


Figure 36: ROC curve estimation from mass of ML1 trained in scaled MC for the lambda particle.

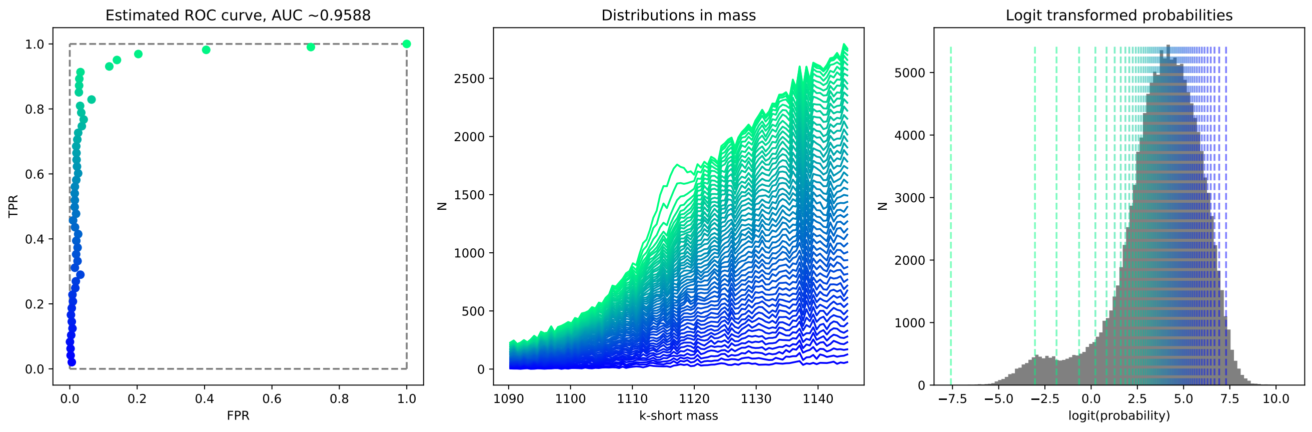


Figure 37: ROC curve estimation from mass of ML2 trained in scaled MC for the lambda particle.

7.2.6 Displaying Decision Boundaries in Higher Dimensions / Decision Boundaries for Correlated Features

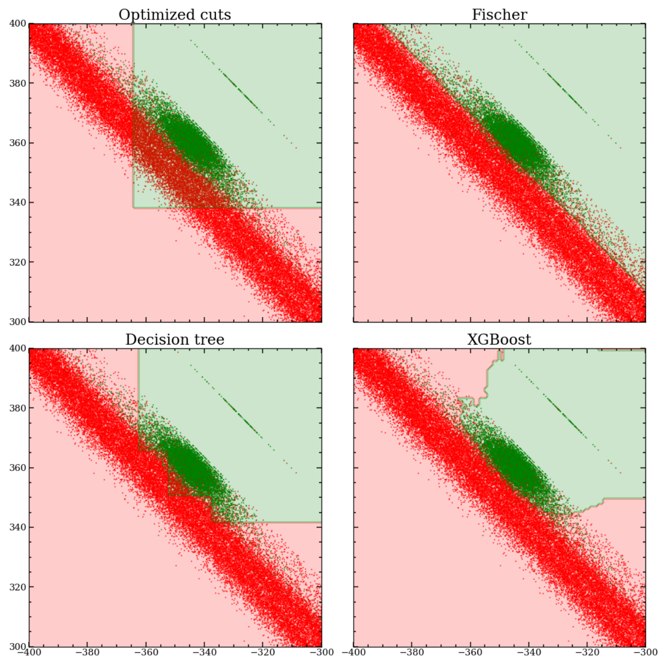


Figure 38: Decision boundaries for correlated features. Here the two features from figure 3 are rotated by 45 degrees into each other, resulting in a pearson correlation of about -1.

7.2.7 ROC-curves for XGBoost in MC

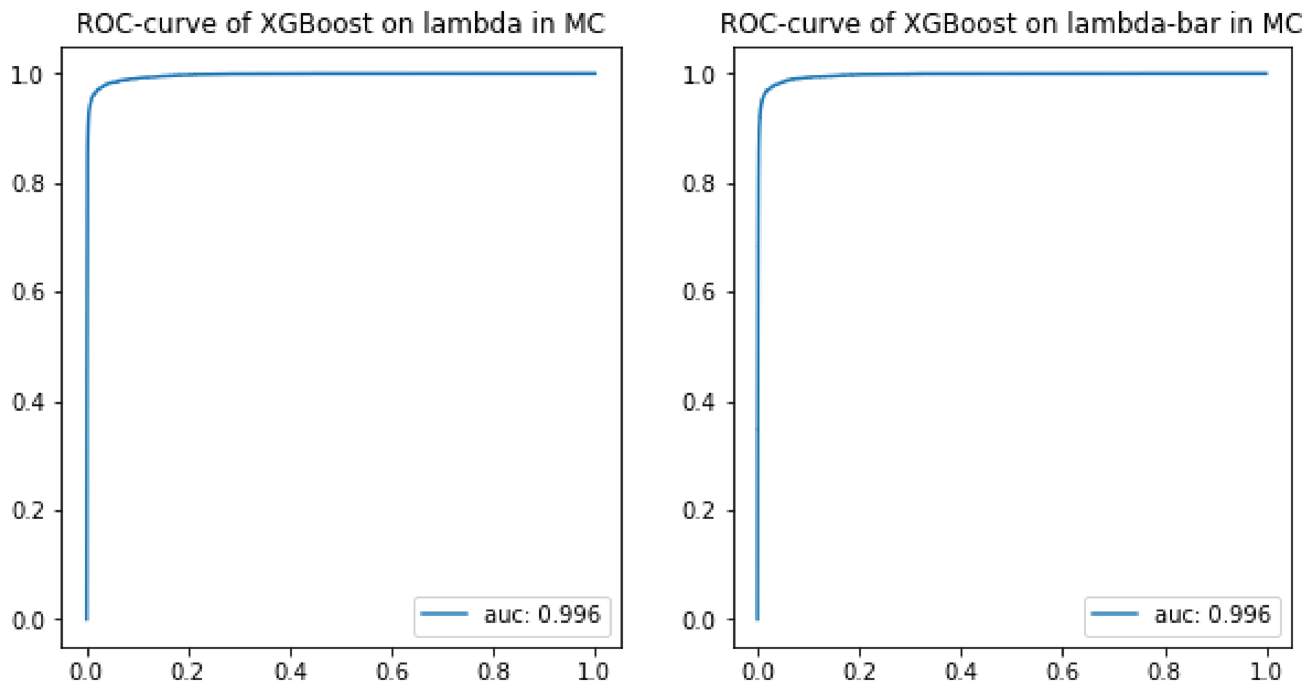


Figure 39: ROC curves for XGBoost trained in MC and tested in MC.

7.2.8 Fitting the Uncertainties

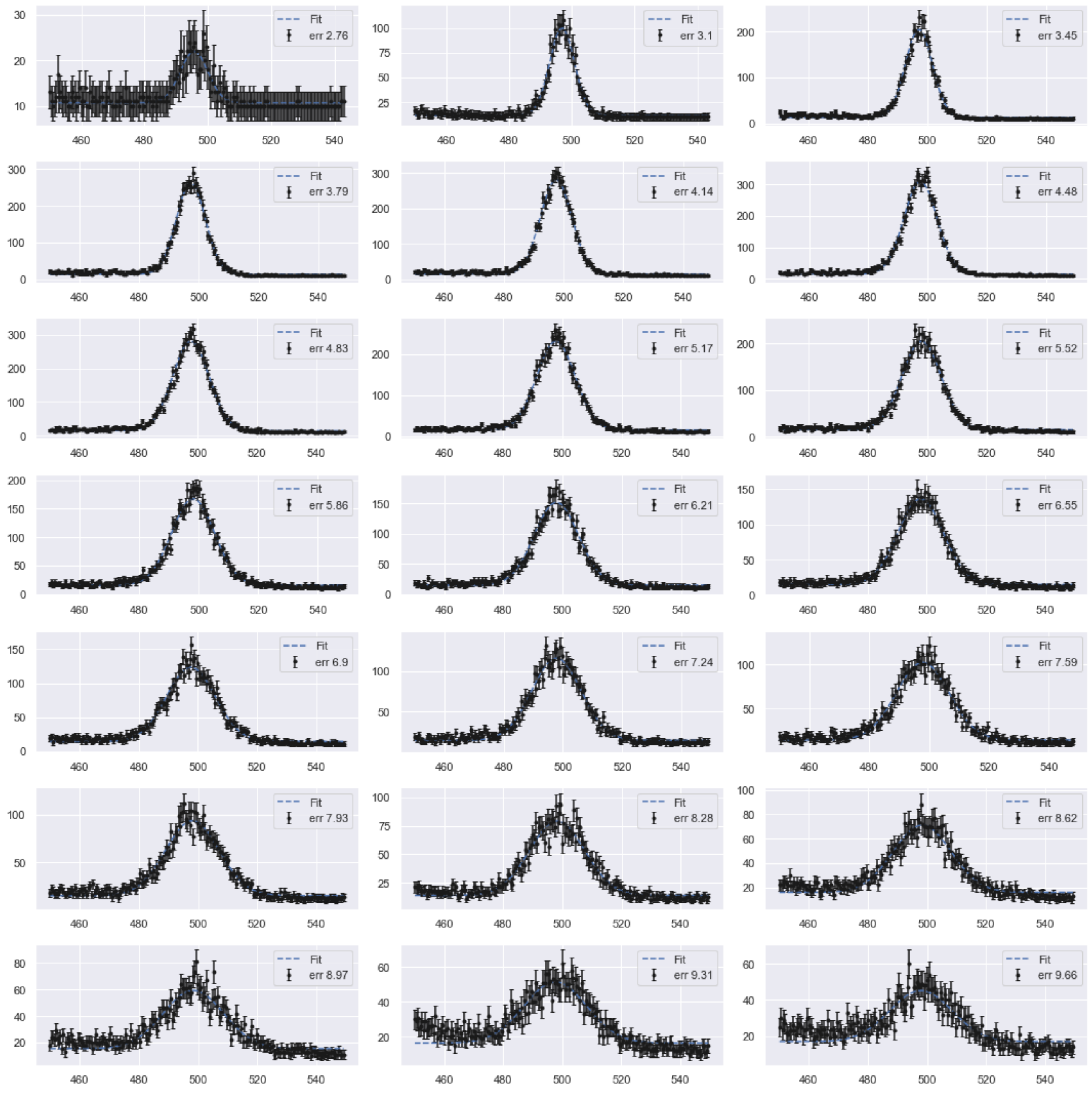
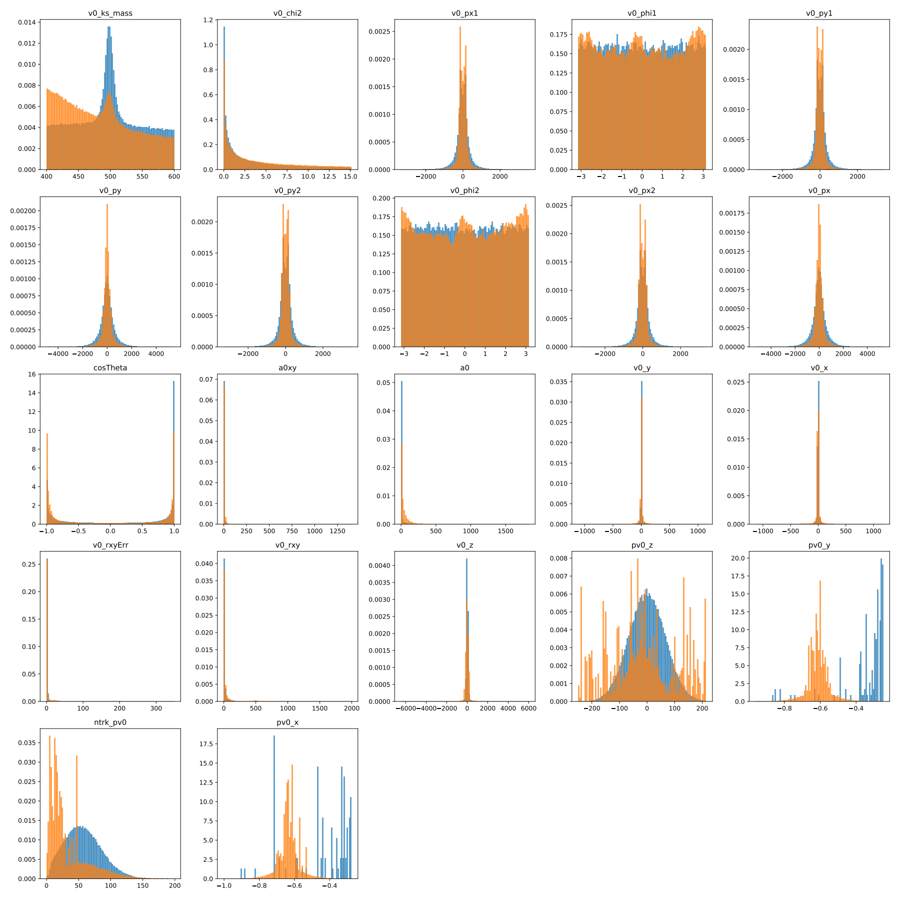
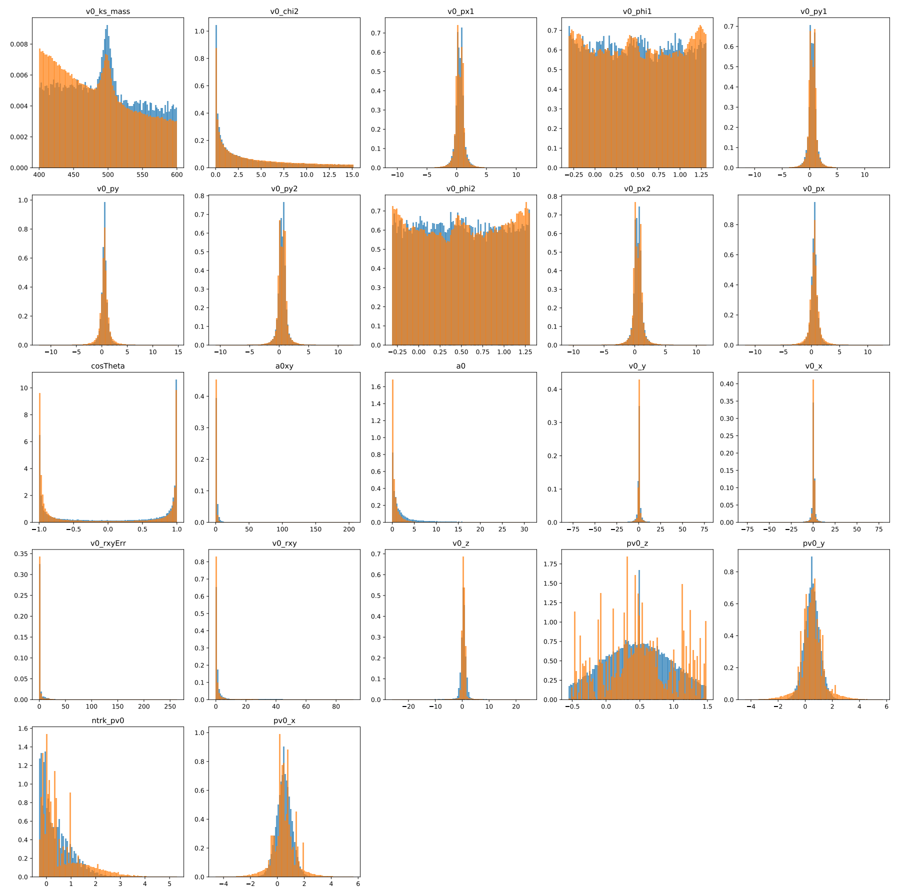


Figure 40: The fit for every point in figure 5.



(a) Original distributions



(b) Scaled and reweighed distributions

Figure 41: Comparison of original and processed (scaled and reweighted) distributions across the mass-uncorrelated parameters

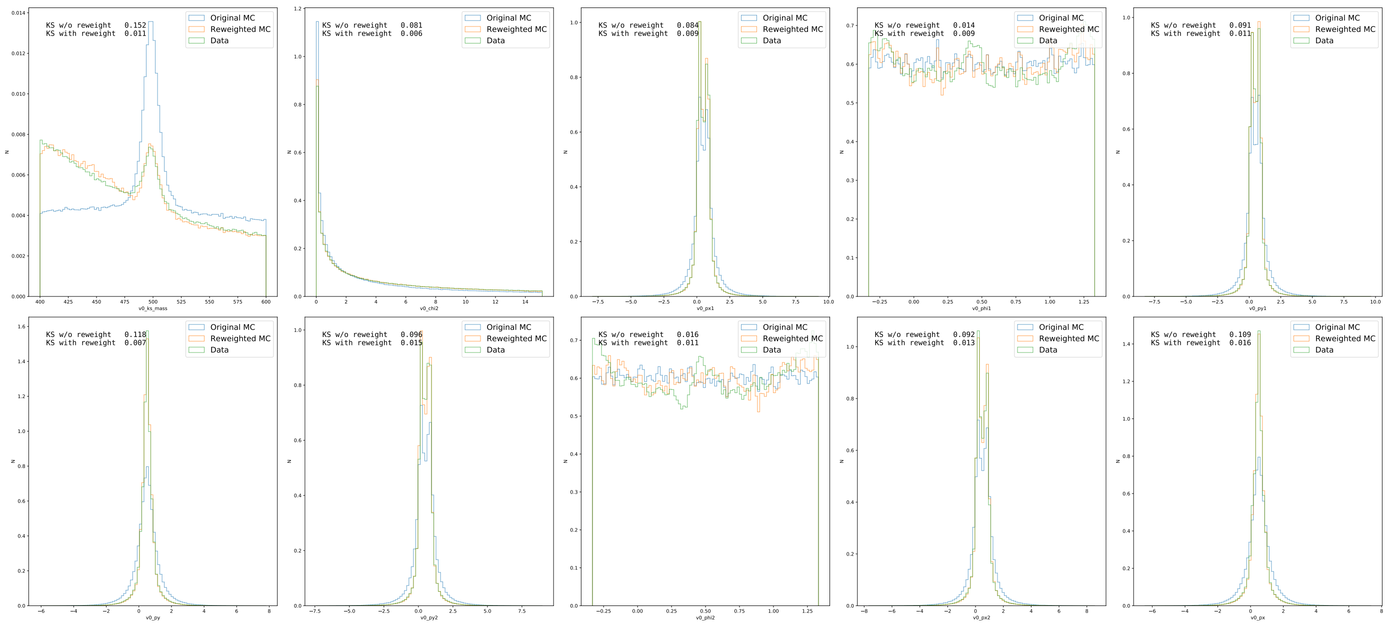


Figure 42: Scaled and reweighted distributions for ML1

Correlations between K_S^0 predictions from models trained in MC and applied in data

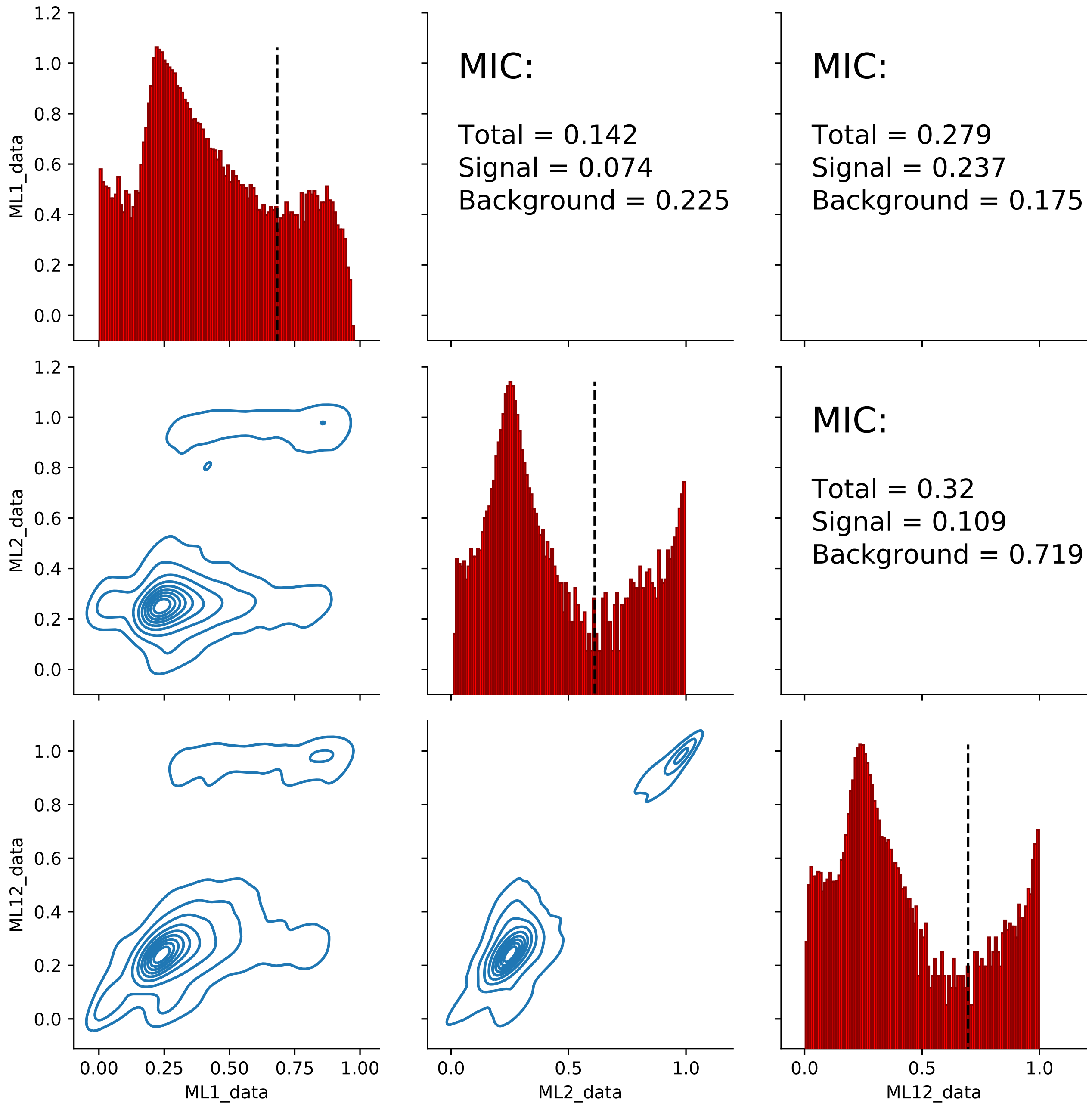


Figure 43: MIC-correlation between different model outputs. ML1 and ML2 were trained on different sets of points of equal size, and ML1+2 were trained on both. All were tested on the same points. On the diagonal, probability density distributions with log(N) on the y-axis is drawn with a line to indicate the background/signal split. In the lower half, contour plots for all prediction scores are plotted as well as classified background (orange) and signal (green). Background and signal is taken as points that both models classify as belonging to either class. In the upper half, summary MIC-scores for all scores, background only and signal only are shown

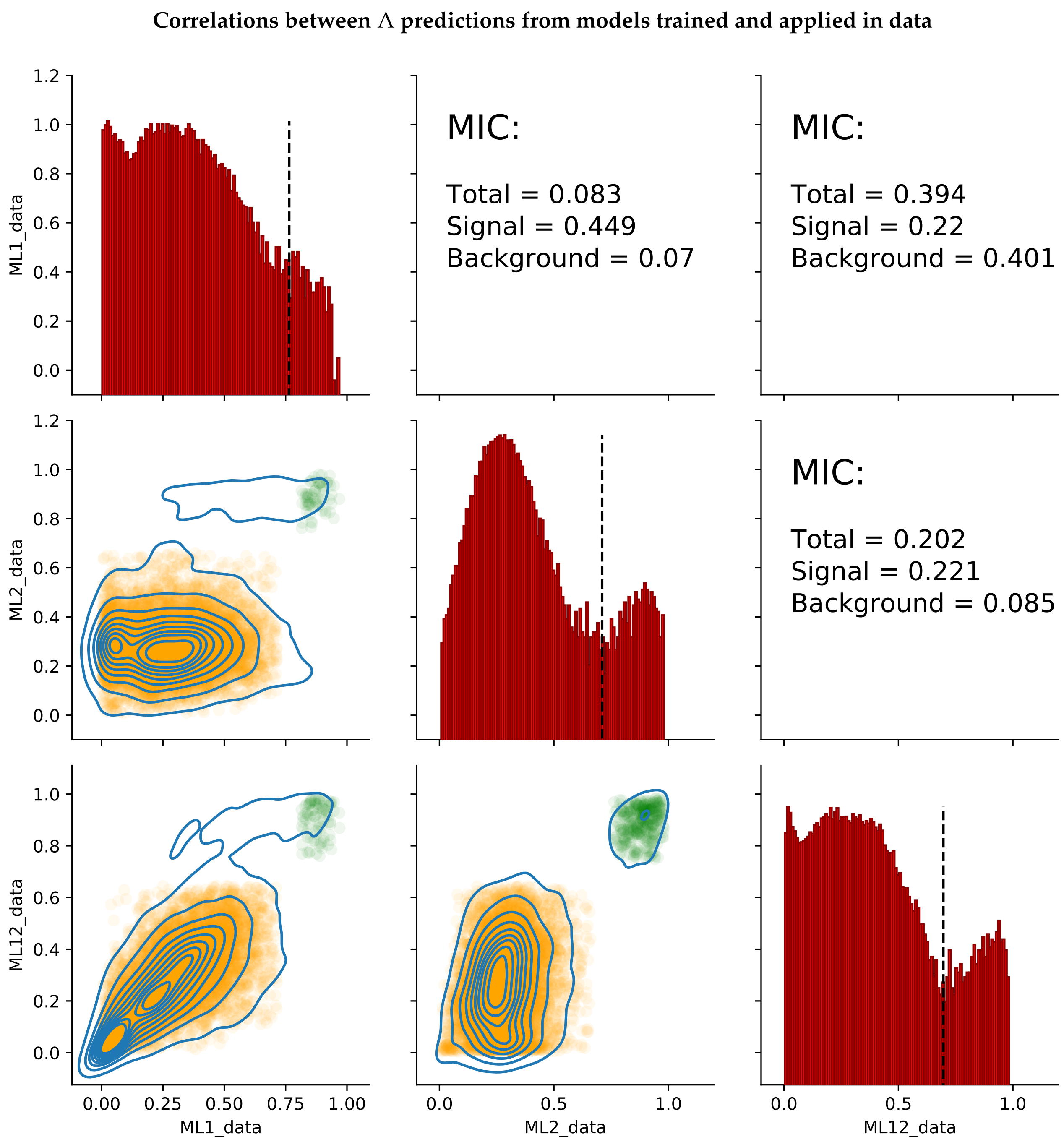


Figure 44: MIC-correlation between different model outputs. ML1 and ML2 were trained on different sets of points of equal size, and ML1+2 were trained on both. All were tested on the same points. On the diagonal, probability density distributions with log(N) on the y-axis is drawn with a line to indicate the background/signal split. In the lower half, contour plots for all prediction scores are plotted as well as classified background (orange) and signal (green). Background and signal is taken as points that both models classify as belonging to either class. In the upper half, summary MIC-scores for all scores, background only and signal only are shown

References

[1] Regents of the University of California. Particles data group. https://pdg.lbl.gov/.

[2] Ronald A Fisher. The use of multiple measurements in taxonomic problems. Annals of eugenics, 7(2):179–188, 1936.

[3] XGBoost developers. Introduction to boosted trees. https://xgboost.readthedocs.io/en/latest/tutorials/model.h

[4] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. arXiv e-prints, page arXiv:1603.02754, March 2016.

[5] Abhishek Sharma. What makes lightgbm lightning fast? https://towardsdatascience.com/what-makes-lightgbm-li

[6] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 4765–4774. Curran Associates, Inc., 2017.

[7] David N. Reshef, Yakir A. Reshef, Hilary K. Finucane, Sharon R. Grossman, Gilean McVean, Peter J. Turnbaugh, Eric S. Lander, Michael Mitzenmacher, and Pardis C. Sabeti. Detecting novel associations in large data sets. Science, 334(6062):1518–1524, 2011.

[8] hep-ml developers. hep-ml github. https://github.com/arogozhnikov/hep_ml.

[9] Particle Data Group et al. Review of Particle Physics. Physics Letters B, 667(1):1–6, September 2008.

[10] Justin Stevens and Mike Williams. uBoost: A boosting method for producing uniform selection efficiencies from multivariate classifiers. arXiv e-prints, page arXiv:1305.7248, March 2016.

[11] Laurae. Parameter matching between xgboost / lightgbm. https://sites.google.com/view/lauraepp/parameters.