Review guidelines for ML-based analysis in the CLAS Collaboration

Richard Tyson, Pierre Chatagnon

October 30, 2026

Abstract

At the time of writing, no outright decision has been made to enforce any particular guidelines in analysis reviews. This document provides suggestions on the requirements for reviewing machine learning–based analyses, with the goal of facilitating the process for both analysers and reviewers.

Machine learning has become more and more prevalent in high-energy physics, including at CLAS12 which has seen an increase in the use of machine learning with diverse applications from online and offline reconstruction to the interpretation of physics results. These applications of machine learning at CLAS12 have typically had positive impacts on the physics output of the collaboration. However, the use of machine learning in physics analyses has complicated the review process. The present guidelines aim to prevent the introduction of biases in the physics outcome of CLAS12 analyses, and to facilitate the analysis note review process by introducing a set of clear checks and plots to validate machine learning algorithms.

The document starts with a definition of biases often encountered in the application of AI/ML to physics analyses. The document then sets out:

- A list of best-practice recommendations.
- A list of suggested plots.
- A review checklist.
- Feedback questions on these guidelines for reviewers of analyses with AI/ML components.

To support these guidelines, an introductory user guide to machine learning and a glossary are given in the Appendix (including two examples of CLAS12 ML analysis with their corresponding review checklist). Additional information on the deployment of ML tools at CLAS12 is also provided in Appendix. Finally, the results of the survey used to establish these guidelines is listed at the end of the additional material.

Contents

1 Dataset bias **3**

2 List of best practices **4**

3 List of plots **5**

4 Review checklist **6**

5 Review feedback **7**

A Glossary **9**

B Introduction to Machine Learning **10**
B.1 Quick Guide to Machine Learning . . . 10
B.1.1 General considerations . . . 10
B.1.2 Training and testing dataset . . . 10
B.1.3 Designing and training the algorithm . . . 11
B.1.4 Evaluating the performance of the algorithm . . . 12
B.1.5 Further optimization of the algorithm . . . 14
B.2 Algorithms Galore . . . 14

C Example analyses from CLAS12 **18**
C.1 Example 1 (good analysis): Lepton identification . . . 18
C.1.1 Description of the analysis . . . 18
C.1.2 Review checklist . . . 22
C.2 Example of analysis 2 (bad analysis): J/ψ event selection using dilepton invariant mass . . . 23
C.2.1 Description of the analysis . . . 23
C.2.2 Review checklist . . . 28

D Deploying AI/ML Tools **29**

E Results of the initial survey on ML-based analysis reviews **31**
E.1 Questions . . . 31
E.2 Answers . . . 31
E.3 Other Feedback: . . . 34

1 Dataset bias

In validating machine learning algorithms in CLAS12 analyses, it is critical to understand that good performance does not necessarily imply physical correctness. One of the risks is the presence of **dataset bias**, where the algorithm learns to exploit misleading information in the training data rather than the intended physics. This type of failure is not due to poor training procedures or overfitting in the traditional sense. An algorithm may appear to perform well, even on held-out validation or test datasets if those datasets share the same underlying biases as the training set. This makes such issues hard to detect using conventional performance metrics, and highlights the need for rigorous, physics-informed validation.

Dataset bias arises when the training data does not accurately reflect the true variability of the target distribution. It can manifest in several forms:

- **Spurious correlations:** the algorithm relies on input features that are statistically correlated with the targeted class in the training data but are not causally related to the underlying physics.
- **Representation bias:** the range or diversity of the training samples is insufficient or the training data itself is unrepresentative of the data to which the algorithm will be applied.
- **Sampling bias:** the process of selecting training data systematically favors certain values or ranges of a physics observable, leading to an unbalanced or skewed dataset.

A common example of spurious correlations occurs in particle identification (PID). If one particle type in the training set is measured over a different momentum range than another, the algorithm can learn to associate particle types with momentum. This can happen without explicitly using momentum as an input variable, since many detector subsystem responses are momentum-dependent. In turn, this will lead to incorrect classifications when deployed on data with different momentum distributions.

Representation bias frequently arises when using simulated data for training. Simulation makes it easier to remove certain biases and ensures correct labels, but it may not fully reflect real detector performance. For PID, simulated datasets will contain only the relevant particle types and be perfectly labeled, but may fail to model detector effects accurately. Alternatively, training directly on CLAS12 data, for example by tagging final state particles using specific exclusive reactions, may contaminate the labeling due to background in the reactions. It may also lead to mismatches between the kinematics of the tagged reactions and those in the target use case, again producing unrepresentative training data and degraded performance at deployment.

A particularly problematic case of sampling bias is **observable sculpting**, where the training labels are strongly correlated with a physics observable later used in the analysis. The example given in Appendix C demonstrates how a bias can be set up in the training sample towards an invariant mass that is fitted to measure a cross section. If the algorithm learns to discriminate using that observable, applying selection thresholds on the algorithm output can distort the observable distribution, artificially enhancing peaks or suppressing backgrounds. Aside from the obvious problem of artificially distorting yields in cross section or asymmetry measurements, observable sculpting contaminates statistical inference as the same observable is used both in training and as a final analysis variable. This will notably impact fits to the observable. Observable sculpting must be strictly controlled in physics analyses and can be identified with, for example, synthetic validation sets with known contributions from different event sources (e.g. signal and background) and input/output tests on toy datasets with known observables.

2 List of best practices

Certain recommendations may not apply to certain tasks and some recommendations are redundant. Use your best judgement when going against these recommendations and carefully explain why that is a sensible thing to do.

- Task Selection: Avoid tasks that have more potential to bias observables, due to the input features and training dataset having enough information to correlate classes to physics observables (e.g. for a multi-particle final state, prioritise single-particle PID over event selection).
- Physics-less Simulated Training Data: Avoid simulated training datasets created with given physics processes (e.g. for event selection, avoid multi-particle final states simulated with resonances, prioritise "flat" data).
- Uncontaminated Non-Simulated Training Data: Avoid non-simulated training datasets with large backgrounds or unfold the contributions from backgrounds [1] (e.g. do not create training sets from exclusive reactions with large backgrounds that are poorly understood).
- Feature Selection: Use physically motivated input features. If you are injecting physics in your training dataset, avoid input features that have enough information to extract your physics observables (e.g. for event selection with multi-particle final states simulated with resonances, avoid using input features where these resonances are identifiable).
- Feature Importance: Establish the input feature importance in a dataset [2], ensure the chosen input features are physically meaningful.
- Hyperparameter Selection: Report your algorithm architecture, its hyperparameters and justify the choice of algorithm and hyperparameters.
- Output Variable Correlation to Observable: Estimate the correlation between your algorithm's output and your physics observables.
- Introduce De-Correlation with Observables: The loss functions or training data can be modified to avoid correlations with physics observables [3]. Binning can also help remove the dependence on observables. De-Correlation techniques can make training slower and less likely to converge.
- Training Data Comparison: Ensure that your training data matches the data you apply the algorithm to, including variable ranges.
- Non-simulated Validation Sets: When training on simulated data, validate on both simulated and non-simulated data.
- Synthetic Validation Sets: Create test sets with different contributions from event sources (e.g. varying signal to background levels) to understand failure modes.
  If the input features in your training data are different to those in the data you apply your algorithm to (e.g. different ranges), create synthetic validation sets for various scenarios to understand the impact on the output of your algorithm.
- Input-Output Tests: Inject a known observable into your dataset, check that you are not distorting it by applying the ML-based algorithm. Check on a flat, physics-less dataset that you are not sculpting it.
- Impact on Fits: When fitting observables, check if your algorithm changes your fit parameters (e.g. peak fit width increases when increasing threshold on output).
- Comparison to Conventional Method: If possible, compare your results to ones obtained with a non ML method. Demonstrate where the differences come from.
- Vary Threshold on Output: In classification tasks, vary the threshold on your output to estimate the impact of the choice of threshold on your observables and the tests above.

3 List of plots

The following is a necessary but not always sufficient list of plots recommended to include in analysis review involving ML tools. These plots provide inputs to enforce the guidelines presented in this document. Some plots provide redundant information or are unsuited to given tasks, use your best judgement in choosing what suggestions to ignore and justify why.

- **Inputs:**
  - 1D histograms of input features (by class for classification tasks).
  - Full correlation matrix between all inputs.
  - Separation power for each input variable (for classification tasks).
- **Algorithm Outputs:**
  - 1D histograms of output (by class for classification tasks).
  - 2D correlations between each input and output.
  - 2D correlations between output and relevant kinematic variables (e.g. P, θ, ϕ).
- **Training process:**
  - Loss vs epoch.
  - Overtraining check: train vs test output distributions (per class).
- **Dataset comparison:**
  - Input feature distributions for training/test data vs application data.
  - Algorithm output when applied on training/test data vs application data.
- **Performance metrics** (Classification):
  - Receiver Operator Curve (Signal efficiency vs Background rejection) curve.
  - Metrics vs output threshold.
  - Metrics vs relevant kinematics, at various output thresholds.
  - Input features, kinematic variables, and observables for accepted/rejected events, at various output thresholds.
- **Performance metrics** (Regression):
  - Prediction and truth vs relevant kinematics.
  - Metrics vs relevant kinematics.
  - Resolution (truth - prediction) vs relevant kinematics.

4 Review checklist

The aim of the following checklist is to offer suggestions of questions to ask in order to help identify potential sources of bias in a ML-based analysis. As for the previous list of plots, certain recommendations may not apply to certain tasks.

Review item	Questions	Validation
Task Selection	Does the task require AI/ML ? Can a simpler AI/ML algorithm be used instead ?	Y / N Y / N
Training Data	Is the training dataset representative of the data the algorithm will be applied to? Is the use (or not) of simulated data justified? If simulated, does it have physics in it? If not-simulated, does it avoid contamination?	 Y / N Y / N Y / N Y / N
Feature Selection	Are the input features well motivated ? Was the feature importance evaluated?	Y / N Y / N
Algorithm and Metric definition	Can the algorithm and metrics be understood solely by reading the note ?	Y / N
Hyperparameter selection	Are the algorithm and its hyperparameters explicitly mentioned and justified ?	Y / N
Plots	Are the suggested plots present?	Y / N
Correlations to Observables	Are the correlations between algorithm output and observables evaluated? If needed, were de-correlation procedures applied?	 Y / N Y / N
Validation Sets	Were metrics produced on non-simulated data? Was the algorithm applied to synthetic data? Were input-output tests done?	Y / N Y / N Y / N
Comparison to Conventional Methods	Is the comparison possible and present?	Y / N
Threshold Checks	Is the threshold on the algorithm output varied to evaluate the impact on kinematics, observables and fits?	Y / N

5 Review feedback

These guidelines are expected to evolve as the collaboration's experience and understanding of reviewing analyses with machine learning components deepen. A set of questions addressed to reviewers of analysis notes can be found below. Reviewers of ML-based analyses are invited to send their answers and comments to these questions so that these guidelines can be completed and improved.

Question 1: Please describe the AI/ML tool or analysis that you were asked to review.

Question 2: Did you find this review particularly challenging compared to others without AI/ML components?

Question 3: Did you use these guidelines to review the analysis ? If so, did you find them useful ?

Question 4: Did you notice missing elements that should be included in the guidelines ?

Question 5: If relevant, please describe further questions and studies, beyond those in these guidelines, that you asked for when reviewing the AI/ML component of the analysis.

Question 6: Did you have trouble with understanding:

- The mechanics of the algorithm and technical terms?
- Figures or metrics presented by the analyser?
- The impact of the algorithm on the results?
- How to validate the algorithm?
- What to look for and what might be missing in the analysis?

Question 7: Any further comments?

Additional material

A Glossary

- Algorithm: The machine learning algorithm, often referred to as learning algorithm, model, or by the algorithm type (eg BDT, neural network etc).
- BDT: Boosted Decision Tree
- Classification: Machine learning task aimed at separating events into classes (eg PID where particles are separated by type).
- CNN: Convolutional Neural Network
- Epoch: One training cycle
- Feature: Input variable of a ML algorithm, often also referred to as input, input feature, variable.
- GNN: Graph Neural Network
- Hyperparameter: User defined parameter of the algorithm (e.g. number of neurons in a Neural Network).
- Loss function: Function minimised by the algorithm during training (eg mean squared error).
- MLP: Multilayer Perceptron
- Model: See Algorithm.
- Regression: Machine learning task aimed at predicting one or more quantities (eg predicting the momentum and angles of tracks).
- Response: The output of the algorithm. In binary classification, this is typically the output of the signal class.
- ROC: Receiver Operator Curve (Signal efficiency vs Background rejection curve)
- Test Dataset: A dataset that is held out during training, to validate that the model generalises to unseen data. Also called validation set.
- Threshold: Cut applied on the output of a ML algorithm to separate background and signal

B Introduction to Machine Learning

This section gives a quick introduction to machine learning to supplement the guidelines. More detailed and engaging resources can be found here [4–6], with more thorough and digestible discussions of machine learning than could be achieved here. This section will aim to provide the reviewers with enough material to get a good understanding of machine learning. This section describes the general layout and components of a ML-based project and some of the various algorithms that are often used at CLAS12.

B.1 Quick Guide to Machine Learning

B.1.1 General considerations

Machine learning can be loosely described as a set of tools or algorithms that leverage large multidimensional datasets to create predictive models without having to explicitly code in these models [4].

Machine learning algorithms can be categorised based on how they "learn" and what their objectives are. Most of the machine learning algorithms used at CLAS12 are examples of supervised learning. In this case, the dataset passed to the algorithm during training contains a feature matrix $\mathbf{x}$ and an associated label vector $\mathbf{y}$ with an equal number of rows (one per event) [4]. A ML-algorithm outputs a predicted label vector $\mathbf{y}'$ [1] using a combination of its parameters and the corresponding row of the feature matrix $\mathbf{x}$ [4]. The feature matrix contains variables, known as input features, used by the algorithm to make its prediction. The training process aims to adjust the parameters of the algorithm in order to minimize a loss function which quantifies the difference between the predicted labels $\mathbf{y}'$ and the truth labels $\mathbf{y}$. Once the algorithm is fully trained, it can be deployed on unlabeled data, where $\mathbf{y}$ is unknown. For the rest of this document, the term *machine learning* is assumed to refer to supervised learning. [2] [3]

As mentioned, the goal of a supervised learning algorithm is to produce predicted labels $=\mathbf{y}'$ for each event (ie each row of the feature matrix $\mathbf{x}$). In practice, the terms *classification* and *regression* are used to distinguish between the cases where the set of values in $\mathbf{y}$ and $\mathbf{y}'$ are discrete or continuous [4]. Classification can be thought of as trying to label each event (i.e. row of $\mathbf{x}$) into a certain number of classes. In other words, the labels $\mathbf{y}$ only have a finite set of values corresponding to the class label. Particle ID is a good example of classification problem where particles are labeled by their type, with each type corresponding to a class.

Regression problems aim to predict a real number associated with each row of the feature vector. For example, machine learning can be used to model the resolution of a given detector. In this case, the algorithm predicts the deviation introduced by the detector from the true four-momentum of a particle. Another example of a regression task is image generation which aims to create new images from a given input.

B.1.2 Training and testing dataset

The first step towards using a supervised machine learning algorithm is to create a dataset on which to train the algorithm. This dataset is typically separated into a training set and a testing set.

Creating the training dataset might be the most important step as ultimately, the algorithm can only learn based on what it is given to train on. It is also relatively easy to introduce some kind of bias into a training set. A famous example [7] is that of an algorithm trained to distinguish between images of huskies and images of wolves. At first glance, the algorithm seems to perform well, even when applying it to a testing set which was created in a similar fashion to the training set. However, when deployed to new images that were not used in the training or testing set, the algorithm did not perform as well. The reason? In both the training and testing sets, the images of wolves always contained snow in the background, and the algorithm was actually trained to recognise snow [7].

[1] Note that each label y is not necessary a number but can also be a vector.

[2] The opposite of supervised learning, unsupervised learning, uses only the feature matrix $\mathbf{x}$ to train the algorithm which then outputs an unspecified label matrix [4].

[3] The terms AI and ML are often used interchangeably. Machine learning is a subset of AI. In this document, both terms can be understood to refer to the same set of methods.

A common example of dataset bias in hadronic physics is seen when trying to train an algorithm to distinguish between two different particle types, which is commonly known as particle identification (PID). A bias is introduced when creating the dataset due to one of the particle types being measured over a different momentum range to that of the other particle type. As the response of a lot of detector subsystems correlates to momentum, the algorithm then learns to distinguish momentum ranges even when not explicitly trained on momentum, and associates a given particle type to a given range. As such the algorithm is biased towards momentum, and when deploying it, it will incorrectly classify particle types based on their momentum instead of variations in the response of detector subsystems to different particle types.

A common way of creating training datasets in hadronic physics is to use simulated data. This makes it easier to remove any biases from the training data, as it is easier to know exactly what goes into the dataset, and it also makes it easier to have representative data. For example, for PID, the training dataset must only contain examples of the particle types that the algorithm must identify, otherwise it is unrepresentative of the problem at hand. To create such a dataset from data taken with a real detector, identifying the particles that will go into the training data must be done with absolute certainty. Whilst this is feasible, it will most often rely on using a specific reaction to identify this particle type [8]. Assuming this can be done for all particle types that are to be identified by the algorithm (over the same momentum ranges), it might not be the case that the kinematics of these reactions will correspond to the kinematics of the events in the preferred use case. As such the training data would be unrepresentative of the task at hand, which might lead to a decreased performance when deploying the algorithm. The drawback from using simulated data is that this may not always be fully representative of data taken with the actual detector, and care must be taken in using input features that are well simulated.

An important aspect of creating a training dataset is the choice of variables, so called input features, that go into the dataset. In the case of particle identification with the CLAS12 detector, there are many subsystems that could contribute meaningful information towards identifying final state particles detected by CLAS12. Including too many input features may hamper the training progress as this can lead to dimensionality issues whereby the algorithm will be slower to train and to make predictions as it needs to treat a larger amount of information. Superfluous input features that do not contain any relevant information for the task at hand can also lead to correlations in the training data that can introduce some bias in the algorithm. In practice, relevant input features can often be selected using appropriate domain knowledge. For example, some CLAS12 subsystems were built to identify specific particle types and would not be useful to identify other particle types. Feature selection techniques can allow to rank input features with the aim in mind to remove the lowest ranked input features. Dimensionality reduction techniques such as principal component analysis can allow to remove highly correlated input features that contain redundant information, or to construct new input features from highly correlated ones that contain all the relevant information.

The testing dataset is used to check that the algorithm is not overfitting to the training set [4] (i.e. it is not creating a model so specific that it only applies to the dataset used to train the algorithm). This can be checked by applying the algorithm to the testing dataset: if the algorithm performs poorly on the testing set but performs well on the training set then it is overfitting, the expectation being that the model is generalisable to unseen data. Typically the same data is used to create both the training and testing sets.

B.1.3 Designing and training the algorithm

The next step is to choose and train a ML algorithm. There exist a large number of different ML algorithms. Some of the most common are decision trees and neural networks. Certain algorithms are better suited to certain tasks (e.g. autoencoders are known to be effective for image generation tasks). The next section provides details on the different kinds of learning algorithms and their specificities.

In this subsection, we focus on the procedure of training these algorithms. The training process optimises certain variable components of an algorithm in order to optimise their performance at predicting $\mathbf{y}'$. The performance of an algorithm is evaluated using a loss function that typically estimates the deviation between the algorithm's prediction $\mathbf{y}'$ and the true labels $\mathbf{y}$. For example, for a label matrix $\mathbf{y}$ of size $N \times 1$ which has one label to predict for each of N predictions, and an associated prediction matrix $\mathbf{y}'$ the mean squared error (MSE) is defined as:

$$MSE = \frac{1}{N}\Sigma_i^N(\mathbf{y}_i - \mathbf{y}_i')^2 \ . \tag{1}$$

The *MSE* loss is often used for regression tasks where the output of a learning algorithm aims to reproduce a continuous variable.

For classification tasks, the cross entropy loss compares the predicted probability distribution of a model to the true probability distribution. The binary (i.e. two classes) cross entropy loss is shown below:

$$L(f(x_i)) = -\sum_i (y_i \log f(x_i) + (1-y_i)\log(1-f(x_i))) \ . \tag{2}$$

The parameters of an algorithm are optimised to minimise the loss, with the optimisation happening at different intervals depending on the algorithm but such that each element of $\mathbf{x}$ has at least one opportunity to contribute to the optimisation. These intervals are typically called epochs. Figure 1 shows the example of a loss function evaluated at each epoch on a training and a testing set. The loss decreases before reaching a plateau, indicating that the algorithm has converged and finished training for both the training and testing datasets. However the difference between the loss evaluated on the testing and training is indicative that the algorithm is slightly overfitting. This is not a major concern as the loss on the testing dataset does not further increase with additional epochs which would denote that the performance of the algorithm is getting worse as the algorithm trains for additional epochs.

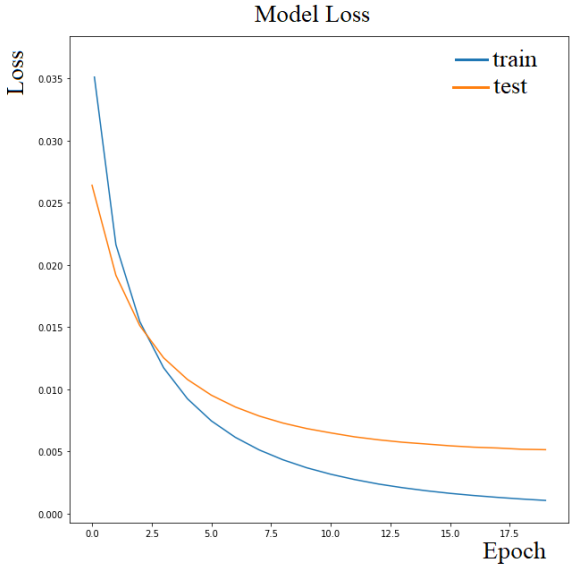


Figure 1: The *MSE* loss evaluated at each epoch on training and testing datasets. The training and testing losses can be used to check for overfitting. As the testing loss is here slightly worse than the training loss, this is a sign that the model is overfitting to the training data as it performs less well on unseen testing data. However, this is not too concerning here as the testing loss does not start to increase again which would demonstrate the model performance becoming worse on unseen data as it trains for longer.

B.1.4 Evaluating the performance of the algorithm

A set of metrics needs to be defined to quantitatively estimate the performance of a machine learning algorithm. In classification tasks, a ML algorithm will typically output a number that can be understood as the probability for a given object/event to belong to a class. The output of a machine learning algorithm is often called the response. As the response of a learning algorithm for each class will not be exactly zero or one but will lie somewhere in between, a threshold placed on the response allows to classify an event as belonging to a class or not. For example, for a task with two classes, the true label for class 1 (often called positive class) will be [0,1]. For a well classified positive event, the response could be [0.1,0.9]. For a badly classified positive, this may be [0.9,0.1]. Some event can have feature not distinctive enough, and this could be lead to label such

as [0.4,0.6]. In this case, placing a threshold at 0.5 on the response of the positive class would allow to select two out of three positive events. However, the true label for class 0 (often called negative class) will be [1,0]. Given three negative events, with responses [0.9,0.1], [0.6,0.4], [0.1,0.9], a threshold at 0.5 on the response of the positive class would select one negative event. Depending on the task at hand, one might want to optimise the negative class rejection, or optimise the positive class selection. [4]

A decision must be made as to where to place the threshold on the response. To exemplify the importance of this, let us consider briefly the ML trigger project described in Ref. [9]. The aim of the trigger is to select events with an electron in a certain region of the CLAS12 detector and filter out those events that do not contain an electron. The positive and negative classes are then defined as: the event has an electron or the event does not have an electron. To determine whether or not an event should be selected by the trigger, a threshold is placed on the response of the learning algorithm forming the trigger. An event with a response above the threshold is kept whilst an event with a response below the threshold is rejected. A confusion matrix can then be created as in Table 1 [9]. The columns correspond to the true columns whereas the rows corresponds to the label assigned by placing a threshold on the response. A True Positive (TP) is defined as an event with an electron that has been correctly selected by the trigger. A False Positive (FP) is then defined as an event which does not contain an electron but was incorrectly selected by the trigger. A False Negative (FN) is an event with an electron which was incorrectly rejected by the trigger. A True Negative (TN) is an event without an electron that was correctly rejected by the trigger [9].

Confusion Matrix	Electron in Event	No Electron in Event
Selected by Trigger	True Positive (TP)	False Positive (FP)
Rejected by Trigger	False Negative (FN)	True Negative (TN)

Table 1: The electron trigger confusion matrix. Table taken from [9].

Metrics such as the purity (P), efficiency (E) and accuracy (A) can then be calculated from the entries in the confusion matrix:

$$E = \frac{TP}{TP+FN} \tag{3}$$

$$P = \frac{TP}{TP+FP} \tag{4}$$

$$A = \frac{TP+TN}{TP+FP+FN+TN} \tag{5}$$

A high efficiency is achieved by selecting most events with at least one electron. Mis-classifying events without an electron in one of the six sectors, for example mistaking other particle types for electrons, decreases the purity of the trigger. Typically a balancing game can be played between both quantities as one decreases as the other increases, when, for example, easing or tightening the requirements on electron identification. This is reflected by the overall accuracy which is highest when maximising the product of the purity and efficiency. Figure 2 shows in the top histogram the response of the learning algorithm for events in the testing set that contained or not an electron, respectively in blue and red. In the bottom graph of Figure 2 the metrics defined above are plotted at different thresholds on the response.

[4] Note that in a binary classification problem where there are two classes, the response for each class is typically equals to one minus the response in the other class. One can only consider the response in one class as the response in the second class would only provide redundant information. Preference is usually given to the response in the positive class, on which the threshold is placed.

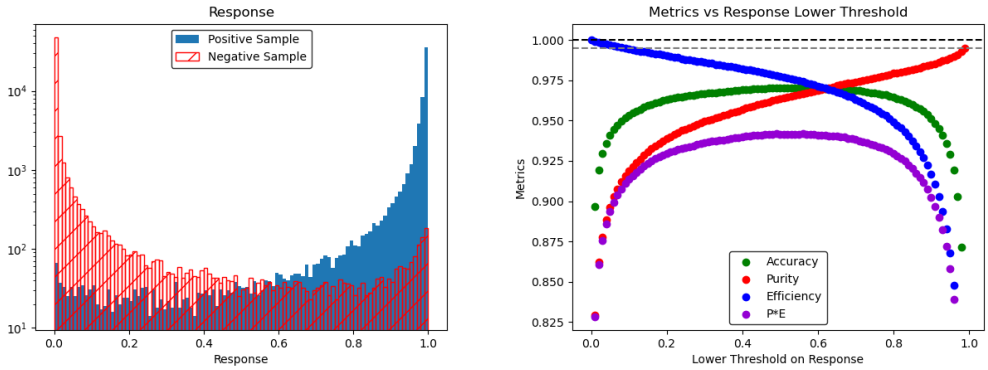


Figure 2: Top: The response of the learning algorithm forming the deep learning trigger. In blue the response for the events of the testing set with an electron, in red that for events without an electron. Bottom: The accuracy, purity, efficiency and the product of the purity and efficiency as a function of the threshold on the response. Plots taken from [9].

The second plot of Figure 2 can then inform the decision making in choosing where to place a threshold on the response. In the case of a trigger, the preference is generally given to maximising the efficiency by placing a threshold at low values of the response, as ideally the trigger should preserve all good data, at the expense of introducing some background. However, in a scenario where good data was cheap and easy to produce, one might choose to maximise the purity by placing a cut at a high value of the response so as to remove most of the background, which will make analysing the data easier. Maximising the accuracy would then achieve a sweet spot in maximising both the purity and efficiency.

B.1.5 Further optimization of the algorithm

The first, second and third steps in setting up a machine learning application were first to produce the training data, second to train the algorithm and third to evaluate its performance on a testing dataset. The next step is to optimise what are known as the algorithm's hyperparameters. These are user defined components of the algorithm, such as the number of layers or nodes in a neural network. More detail will be given in the next section on what these hyperparameters are for different algorithms. Typically the second, third and fourth step will be repeated a few times until the performance of the algorithm is deemed satisfying, or as high as achievable. Note that the third step, evaluating the performance of the algorithm, should also include detailed validation in order to study any potential bias introduced into the analysis. Once all the steps are completed and the algorithm is fully validated and as good as it can be, it can then be deployed.

B.2 Algorithms Galore

There are many different kinds of supervised learning algorithms that have been developed over the last thirty years as machine learning has become more and more prevalent in modern computing practices. In fact, there are too many algorithms to describe here without completely derailing the narrative of this document. As such, emphasis will be placed on the algorithms commonly used at CLAS12. In particular, these comprise decision trees and neural networks, and variations on these.

Decision trees (DT) [11, 12] are perhaps, conceptually, the simplest algorithms to describe. DTs, exemplified in Figure 3, are composed of nodes which branch out into child nodes. The last node at the end of a branch is known as a leaf and returns the predicted label. DTs attempt to split the training data by repeatedly splitting it at each node. The splits are performed by selecting simple cuts on a random choice of input features such that a specified loss function is minimised. One example of such a loss function is the Gini index G [11]:

$$G = 1 - \Sigma_i \left(\frac{p_{mi}}{p_m} \right)^2 \tag{6}$$

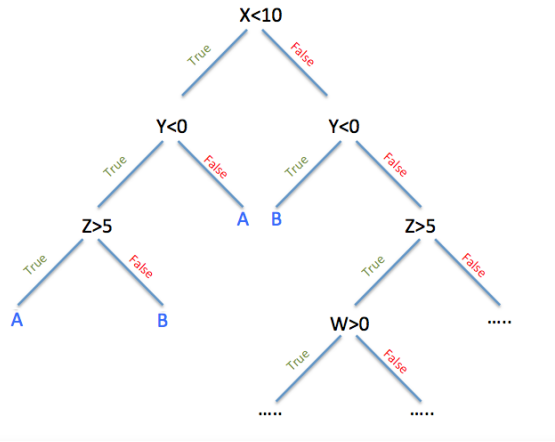


Figure 3: An example of a decision tree, with example cuts on variables W, X, Y, Z at each node, leading to a classification into classes A or B. Adapted from [10].

where p_{mi} is the proportion of samples at node m belonging to class i. The splitting continues until each leaf contains one sample, or a specified maximum depth is attained, where the depth corresponds to the number of nodes in a branch. One common issue with decision trees is that these are prone to overfitting the training data [12]. For example, with an arbitrarily large maximum depth, the splitting can continue to produce a tree that will assign the right label to all entries in the training sample due to very specific cuts that can only correspond to the training sample and will not generalise to another sample.

One simple solution to reduce overfitting and generally improve the performance of decision trees is so called boosting. The idea behind boosting is that it is generally easier to train several smaller models than a single large model whilst still avoiding overfitting. Several different boosting algorithms, such as adaptive boosting (AdaBoost) [13] or gradient boosting (GradBoost) [14] will perform what is known as the boosting algorithm, but the overall set of steps required for boosting remains the same. Initially, the boosting algorithm will assign equal weights to each element of the training sample. A first DT is trained on the training sample and makes a prediction for each event. The boosting algorithm then assesses the model prediction with a given loss function, and increases the weight of elements of the training sample which have a larger error. A weight based on the DT's overall performance on the whole training sample is also assigned to the DT. The weighted data is then passed to the next DT, with the training influenced by the weight on each element of the training sample. As such, subsequent DTs will focus on elements that were poorly predicted by previous DTs. The boosting algorithm will then continue to train DTs, assign weights to the training data and pass the weighted training data to subsequent DTs until the training error reaches a certain threshold, or the number of DTs reaches a user defined maximum. The number of DTs can be loosely related to the number of epochs mentioned in the previous section, as each new DT being trained corresponds to a new optimisation process of the learning algorithm. At prediction time, the weighted mean of the prediction of each DT is then used as the final output. The weight used for each individual DT is the overall weight that was assigned to the DT based on its performance on the whole training sample. In such a way, boosting creates a single strong DT out of an ensemble of weak DTs, with the final algorithm generally referred to as a boosted decision tree (BDT).

The other main class of learning algorithm used at CLAS12 are so called neural networks (NNs) [16]. An example is given in Figure 4. NNs are composed of layers of nodes, often called neurons due to the origins of NNs based on creating learning algorithms that learn as a human brain does. Neurons are interconnected with the output of each neuron in a given layer passed onto each neuron in the subsequent layer. NNs will have an input layer which has as many neurons as there are input features in the feature matrix $\mathbf{x}$ and an output layer with as many neurons as columns in the label matrix $\mathbf{y}$. In the example given in Figure 4, each of the five columns of $\mathbf{x}$ is called x_i. The input of each neuron j in the hidden layer a_j is given by:

$$a_j = f(\Sigma_i(w_{ij}x_i + b_i)) \tag{7}$$

where w_{ij} is the weight of the neuron i in the input layer going to the neuron j in the hidden layer, b_i is the bias

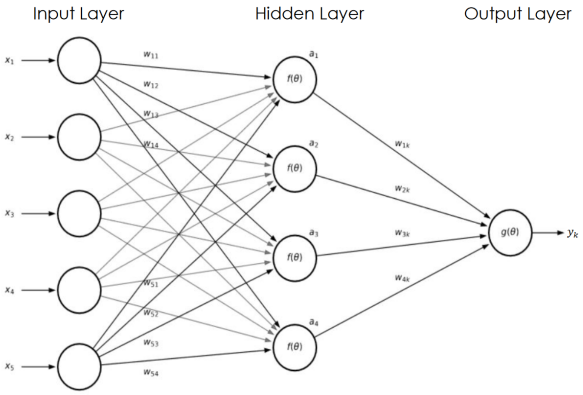


Figure 4: An example of a multilayer perceptron, with one hidden layer, five input nodes corresponding to five input features and one output node corresponding to one predicted label. Adapted from [15].

of the neuron i in the input layer and f is the activation function used in the hidden layer. The output of each neuron k in the output layer is then:

$$y_k = g(\Sigma_j(w_{jk}a_j + b_j)) \tag{8}$$

where the symbols have a similar meaning. The activation function is used to constrain the output of each neuron between zero and one. Typical examples are the tanh function or the ReLu function defined as $R(x) = max(0,x)$. Typically in binary classification tasks the activation function of the output layer is the sigmoid function to obtain the probability that the element of the training sample belongs to class k, with the probability for all classes should sum to 1. For regression purposes the activation function of the output layer is often linear to avoid any transformation of the output of the last node which may make it harder for the NN to train. NNs are then trained by using variations of Stochastic Gradient Descent algorithms, so called optimiser algorithms, that aim to find the minima of the loss function as a function of the weights w_{ij} of each neuron in each layer of the NN. Popular choices of optimiser algorithms are ADAM and ADAMAX [17]. The weights are then changed using a backpropagation algorithm such that $\Delta w_{ij} = -\eta \frac{dL}{dw_{ij}}$ for L the loss function and η the learning rate which can be seen as a step size in weight space.

Figure 4 shows an example of a multilayer perceptron (MLP) [18] which are the most basic type of neural networks. Another type of NN is a convolutional neural network (CNN) [19,20]. CNNs have become the go-to NNs for image classification and more generally computer vision. They are composed of convolutional layers that enhance or remove input features of their input images, and a deep neural network which classifies the output of the convolutional layers. Ref. [9] describes how CNNs can be used to create a deep learning based electron trigger for CLAS12. Autoencoders [21] are a third type of NN, often used for image generation and denoising purposes. The idea is that a first network called an encoder will compress the feature space to a smaller latent space representation. A second network will then decode the latent space representation back to the feature space. This allows for the network to learn the important aspects of the feature space which are encoded into the latent space. By removing the encoder, random entries can be passed to the decoder perpetrating to mimic the latent space representation, leading to the generation of entries in feature space which resemble what the autoencoder was trained on. Autoencoders often combine convolutional layers with a typical neural network and are then called convolutional autoencoders. Finally, Graph neural networks (GNNs) are a type of neural network designed to work with data represented as graphs. They are often beneficial as they allow for variable sized input or output datasets and excel at creating relationships between data inputs.

As mentioned at the end of the previous section, for all types of learning algorithms, some parameters of the algorithm which are not optimised by the neural network are left to be defined by the user. These are typically referred to as hyperparameters. For BDTs, these can be for example the number of trees, the maximum depth of the trees, the loss function, the boosting algorithm or the minimum number of training events required to pass through each node. For NNs, the hyperparameters are for example the number of nodes, the number of neurons in each node, the optimiser, the loss function in each layer, the learning rate, or the number of

training epochs. What hyperparameters are available for tuning will typically depend on the software used to implement the learning algorithm. In python, two very common software packages to train and deploy NNs are *tensorflow* [22] and *pytorch* [23], with the *scikit-learn* [24] package being very useful to train and deploy DTs and MLPs. In Java, the *Deep Java Library* [25] package can train and deploy NNs, including NNs trained with *pytorch*, whilst the Java Graph NN library [26] is useful to train and deploy graph neural networks. In C++, the *ROOT TMVA* [27, 28] package allows to train and deploy DTs, MLPs and NNs.

C Example analyses from CLAS12

C.1 Example 1 (good analysis): Lepton identification

C.1.1 Description of the analysis

This section will discuss the refinement of the electron (positron) particle identification (PID). There are many examples of good usage of machine learning in CLAS12 but this work was chosen here as it highlights all necessary steps taken towards deploying a machine learning algorithm at CLAS12. This work has been detailed in a more complete CLAS note [29] and so will only be briefly summarised here. Note that for simplicity this will be referred to as an electron identification algorithm here but it is trained and applied in much the same way to positrons.

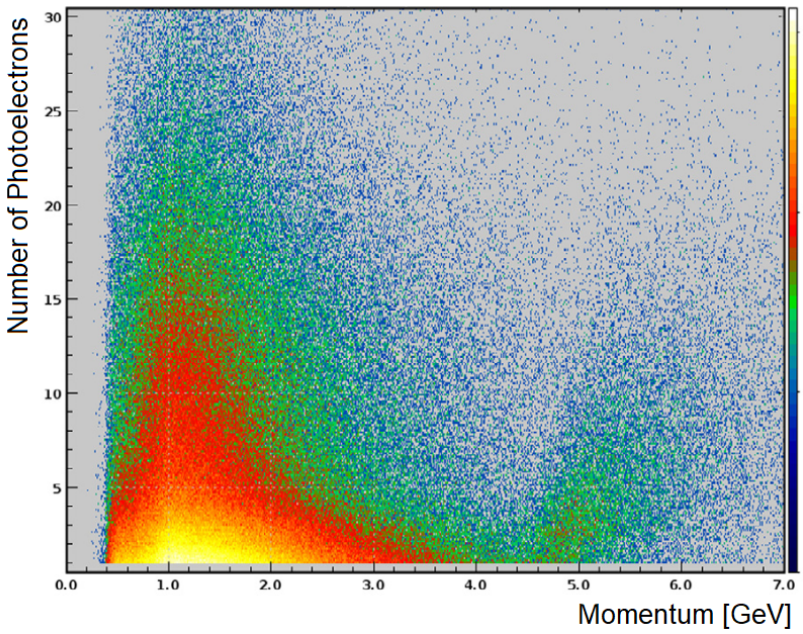


Figure 5: The number of photo-electrons recorded in the HTCC as a function of momentum for positively charged particles. Image taken from [30]

The aim of the method is to remove pion contamination from particles identified as electrons by the event builder PID. The electron event builder PID requires a negatively charged track with an energy deposition above 60 MeV in the PCAL, a sampling fraction around 0.25 where the sampling fraction is defined as the total energy deposited in all calorimeter layers divided by the track momenta, and at least two photoelectrons produced in the HTCC [33]. Figure 5 shows the number of photo-electrons recorded in the HTCC as a function of momentum for positively charged particles. A clear increase is observed above 4.5 GeV. This is due to pions that start to produce photoelectrons in the HTCC above 4.5 GeV. Although pions typically deposit a low amount of energy in the calorimeters, their energy deposition distributions have large due tails due to hadron-hadron showering in the calorimeters. The combination of these two effects leads to pions being mis-identified as electrons by the event builder.

A machine learning algorithm is therefore trained to distinguish between electrons and pions. The ROOT TMVA package is used to do the training, and two algorithms are tried out here, a multilayer perceptron (MLP) and a boosted decision tree (BDT). The algorithms are trained on the particle Momentum (P), the polar (θ) and azimuthal (ϕ) angles, the PCAL, ECIN (Inner Calorimeter), and ECOUT (Outer Calorimeter) sampling fractions (SF), and the sum of the 2nd-moments (M2) for PCAL, ECIN, and ECOUT. To identify any potential bias due to the four-momenta of the final state particles, two versions of the algorithm were trained, one with and the other without P, θ and ϕ. The training sample was created from simulations of single particles, with

electrons for the positive sample and pions that were mis-identified as electrons for the negative sample. All variables in the training sample can be found in Figure 6.

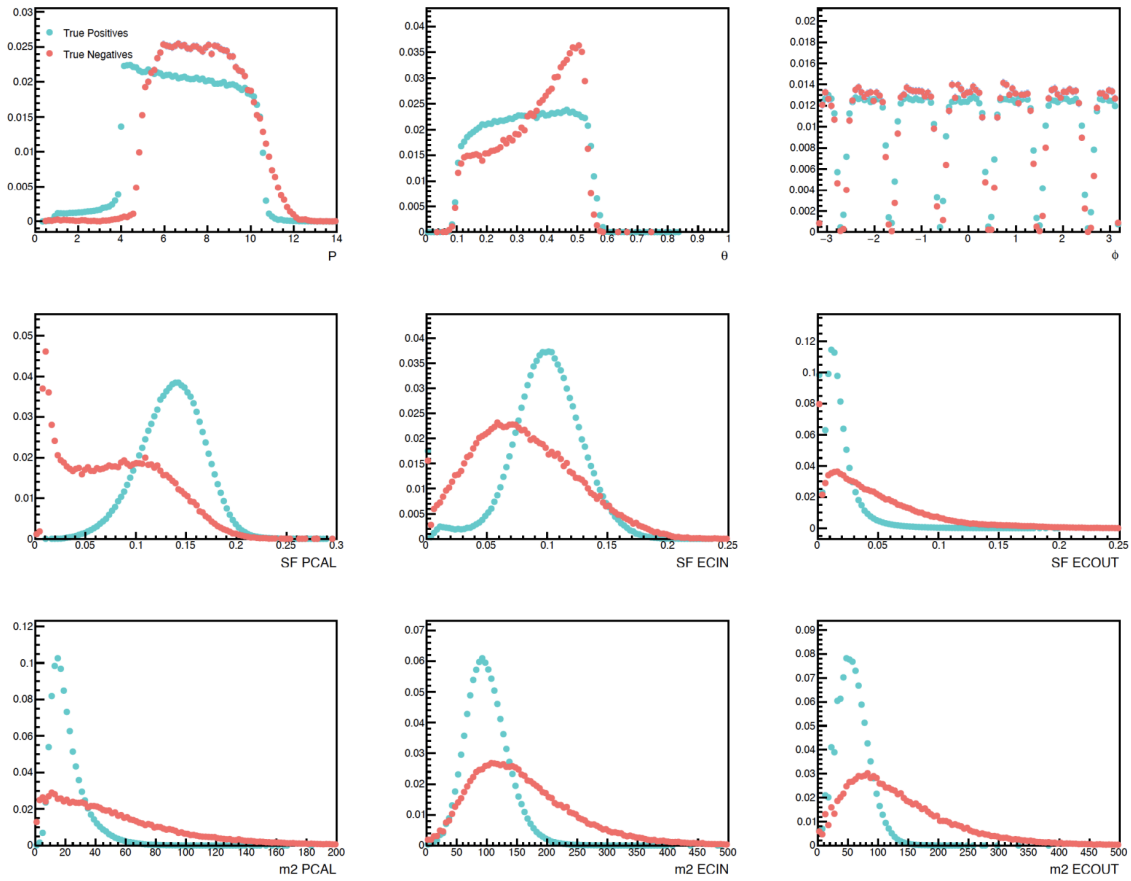


Figure 6: The variables used in the electron PID refinement algorithm, in blue for electrons and red for pions.

The algorithm can then be evaluated on a test dataset created from simulated data using the Receiver Operator Curve (ROC). The ROC curve represents the signal efficiency versus the background rejection (the fraction of the signal kept vs. the fraction of the background removed). An integral of one would denote a perfectly accurate model. Figure 7 shows the ROC curve obtained by the MLP, BDT for both 6 and 9 variables. As shown, the models perform excellently on simulated data, with the 9 variable models outperforming the 6 variable models.

However, the good performance of the model on the simulated data by no means ensures its good performance on CLAS12 data. Errors in creating the simulated training and testing samples could mean that they are unrepresentative of the actual data it will be applied. First, the simulated variables were compared to the reconstructed variables, as shown in Figure 8. Overall there is a good agreement between the variables in simulation and in CLAS12 data, except in the case of the ECIN sampling fraction and M2. Correction factors are then applied to these variables, allowing to find agreement between the simulation and data. Note that the correction does not aim to correct the simulation, but simply to ensure that the data that the algorithm is trained on is representative of the data that the algorithm is applied to.

The algorithm then needs to be validated in CLAS12 data to ensure its good performance. Two channels are used to this end. For the signal, the reaction $e \rightarrow e\gamma$ is used to identify electrons that produce a photon when going through material between the target and detector. A small θ difference between a neutral and a negatively charged particle is a good way to tag true electrons. The PID algorithm is applied to these known electrons to measure how many survive the selection. For the background, the channel $ep- > e'\pi^+(n)$ is used to tag the

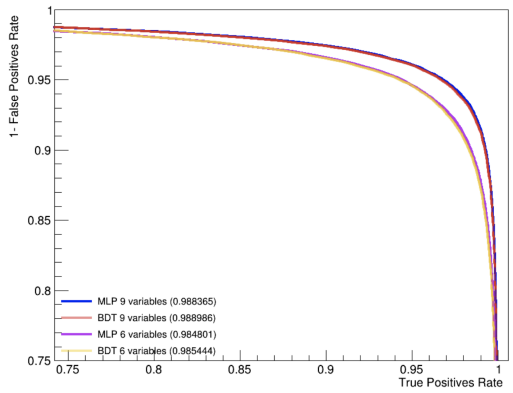


Figure 7: The electron identification algorithm ROC curve when applied to simulated data.

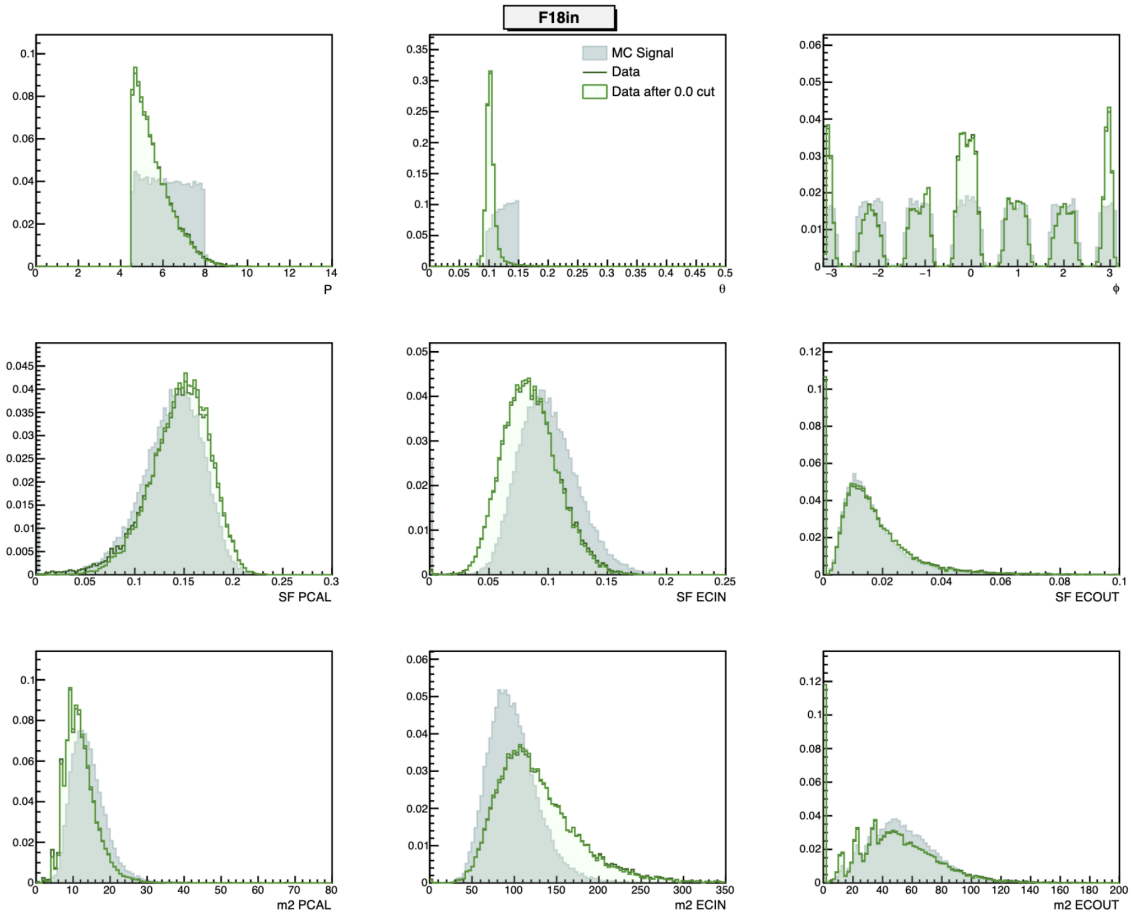


Figure 8: A comparison of the variables used by the PID algorithm in CLAS12 data and simulation.

case where positively charged pions are mis-identified as positrons. Outbending electrons or positive pions are assumed to be similar to inbending positrons or negative pions, and vice-versa, allowing the use of this one channel to validate both the electron and positron identification routines.

The ROC curve resulting from applying the algorithm to CLAS12 data is plotted in Figure 9 and compared to that obtained from simulation. The ratio of the electron efficiency of the algorithm on CLAS12 to simulation data is also plotted in Figure 9, with the ratio being around 1% at the chosen threshold of 0 on the algorithm's

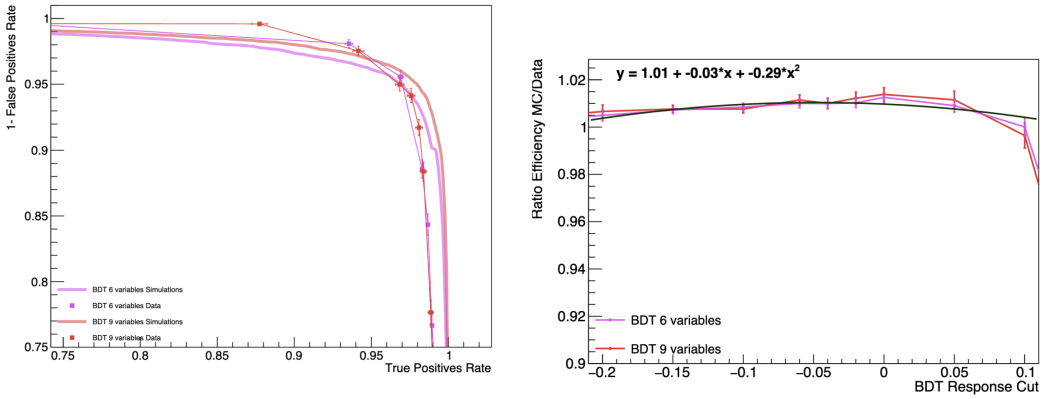


Figure 9: A comparison of the performance of the PID algorithm on CLAS12 and simulated data. The ROC for all BDT models applied to CLAS12 and simulated data is shown on the left, whilst the ratio of the efficiency of the models as a function of the threshold placed on the model is shown on the right.

output. This ratio can then be used to correct the efficiency in simulated data when using the electron PID algorithm, ensuring that the algorithm does not bias efficiency calculations in, for example, cross section measurements.

Overall, the reasons why this application of AI/ML was chosen as an example of a good analysis are the following. First the method is interested in single particle identification, which is less likely to bias physics analyses towards given observables, when these observables are measured with multi-particle final states. Comparing the results obtained by the method with and without the momentum and angles in the training dataset allowed to verify that there is no bias towards specific kinematic ranges. The method then estimated and corrected for the difference in the training sample dataset (simulated) and that to which the algorithm is applied in practice (CLAS12 data). The note [29] does a detailed validation of the method on CLAS12 data, ensuring that the method performs well on the target dataset. Any possible bias on measured cross sections due to different performance on simulated and CLAS12 data is quantitatively estimated by comparing the application of the method to simulated and CLAS12 data. Finally, the work has been written up in a CLAS note, and is being implemented (at the time of writing) in Iguana for accessibility within the collaboration.

We can assess the algorithm by filling out the checklist defined in Table 4. Note that the list of plots was not available at the time when the note was made. The task, electron and positron ID, cannot be made simpler. The use of AI/ML is well motivated as PID is a naturally multivariate task. The data needs to be simulated to create good training samples with cleanly identified final state particles. The training data is either demonstrated to be representative of CLAS12 data, or the differences were corrected when applying the algorithm. There are no connections between potential observables and the input features as this algorithm uses only single particles. The performance of the algorithm was checked on CLAS12 data and compared to the simulated data. The impact of varying the threshold on the algorithm output was verified (and is used as a systematic in analyses using the algorithm presented here). The comparison to conventional methods is estimated, as the algorithms outperform the conventional PID, and further analyses using the algorithms presented here have demonstrated that it does not have an impact on their physics observables. Overall, it is straightforward to verify that the electron and positron algorithms proposed here have no risks of introducing biases into a physics analysis.

C.1.2 Review checklist

Review item	Questions	Validation
Task Selection	Does the task really require AI/ML ? Can a simpler AI/ML task be used instead ?	Yes No
Training Data	Is it representative of the data the algorithm will be applied to? Is the use (or not) of simulated data justified? If simulated, does it have physics in it? If not-simulated, does it avoid contamination?	Needs to use simulation. Training data is representative or corrected. Only single particles.
Feature Selection	Are the input features well motivated ? Was the feature importance evaluated?	Yes No
Algorithm and Metric Definition	Can the algorithm and metrics be understood solely by reading the note ?	Yes Yes
Hyperparameter Selection	Are the algorithm and its hyperparameters explicitly mentioned and justified ?	Mentioned, not justified
Plots	Are the suggested plots present?	Some
Correlations to Observables	Are the correlations between algorithm output and observables evaluated? If needed, were de-correlation procedures applied?	No but obviously not present. No
Validation Sets	Were metrics produced on non-simulated data? Was the algorithm applied to synthetic data? Were input output tests done?	Yes No No
Comparison to Conventional Method	Is the comparison possible and present?	Yes
Threshold Checks	Is the threshold on the algorithm output varied to evaluate the impact on kinematics, observables and fits?	Yes

C.2 Example of analysis 2 (bad analysis): J/ψ event selection using dilepton invariant mass

C.2.1 Description of the analysis

This section will give an example of an AI/ML tool that introduces clearly identifiable bias into an analysis. Here the topic of interest is J/ψ production at CLAS12, where J/ψ decays to an electron positron pair and is identified in their invariant mass by a peak at 3.098 GeV. The J/ψ analyses at CLAS12 typically use cuts on the missing mass squared of $ep \to e^+e^-p(X)$, where the missing particles should correspond to the scattered electron and the missing mass squared peaks at 0 GeV^2. Cuts on Q^2 allow to select quasi-real J/ψ photoproduction, and remove further background. Most of the text and images here have been taken from a 2023 thesis on J/ψ photoproduction on the proton and neutron [31] and more details on the analysis of J/ψ can be found in it. Note that the cross section measurements shown here were very preliminary at the time where the thesis was written and should only be considered in relation to the machine learning application to data analysis.

The machine learning approach aims to specifically select J/ψ photoproduction events, instead of selecting a broader set of di-lepton photoproduction events of which J/ψ photoproduction events are a subset. In this case the training sample is composed of a J/ψ simulation for the positive sample, with the negative sample a combinatorial background created by mismatching particles from different events of the J/ψ simulation. As shown in Figure 11, this ensures that the kinematics of individual particles are roughly the same in both the positive and negative sample. However, as shown in Figure 10, the missing mass squared and the di-lepton invariant mass distributions are different. The learning algorithm will therefore be trained to correlate the kinematics of the individual particles so as to learn to recognise quantities such as the missing mass squared, and the correlation of these quantities and their resolution to the individual particle kinematics.

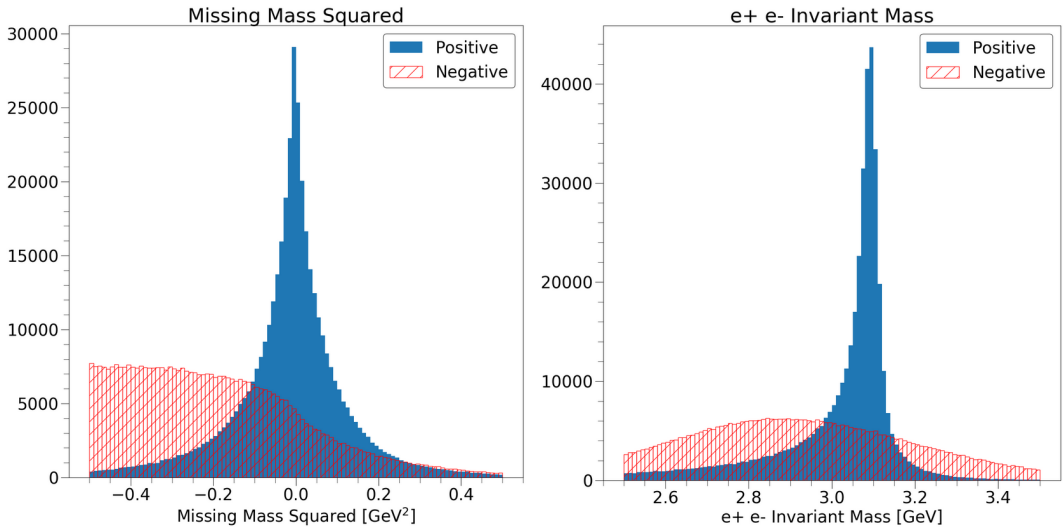


Figure 10: The missing mass squared (Left) and di-lepton invariant mass (Right) for the positive and negative samples of the training data set.

For this use case, a neural network was trained in *tensorflow*. The neural network was composed of five layers, the input layers having nine nodes and a ReLu activation function, the last layer have two nodes and a softmax activation function, with more detail on these activation functions found in Chapter B. The three inner layers had 56, 28 and 14 nodes, each with a ReLu activation function. The neural network used an ADAM optimiser with a learning rate of 10^{-4} and a binary cross entropy loss function. The neural network was trained for 50 epochs. Figure 12 shows the loss evaluated on the training at testing set at each epoch. There is no notable difference between the loss on the training or testing sets indicating that the neural network is not overfitting.

Once trained, the neural network was applied to the testing set to evaluate metrics such as the purity, accuracy, efficiency. The response, and the metrics for various thresholds on the response are shown in Figure 13. Also

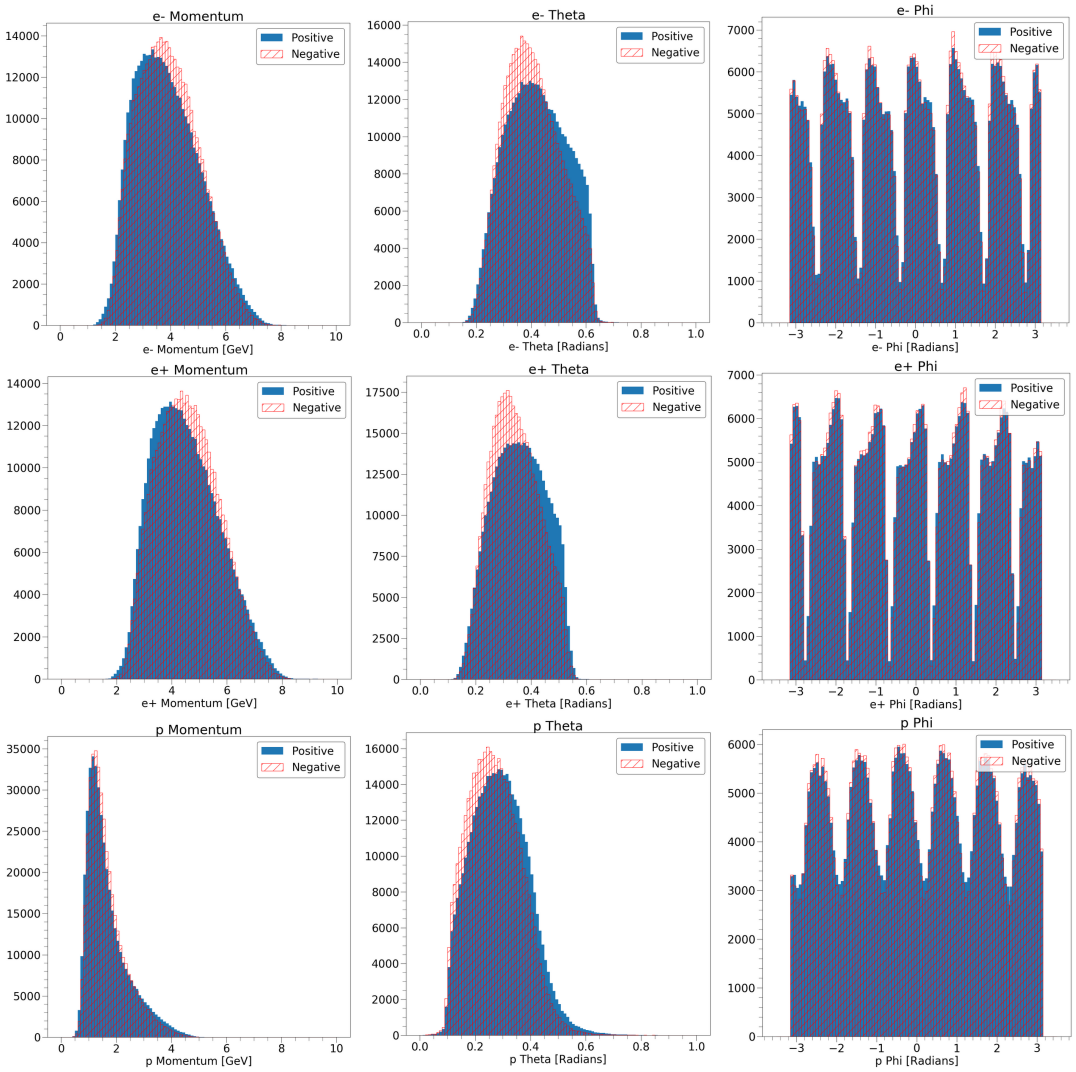


Figure 11: The momentum (Left), polar (center) and azimuthal (Right) angles of the electron (top row), positron (center row) and proton (bottom row) for the positive and negative samples of the training data set.

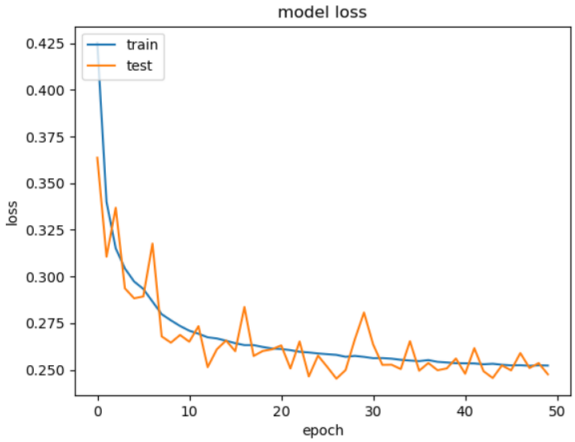


Figure 12: The binary cross entropy loss evaluated at each epoch on the training and testing sets.

shown is the invariant mass distribution of events rejected and selected by at cut at 0.5 on the response. These

are further separated into true negatives and false negatives, which are negative events correctly rejected and positive events incorrectly rejected, and true and false positives which are positive events correctly selected and negative events incorrectly selected. As shown, positive events that are incorrectly rejected occur predominantly in the radiative tails of the J/ψ peak, although some events at the peak were also lost by the neural network. However, the neural network is capable of selecting events in the radiative tails, which suggests that it does not simply select events that have an invariant mass close to the J/ψ mass. Figure 13 also shows the invariant mass distribution of the predicted positives and negatives compared to the true positives and negatives (center). The predicted positives are composed of both true and false positives, with the predicted negatives composed of both true and false negatives. A dip in the predicted negatives compared to the true negatives is observed close to the J/ψ peak, which is accompanied by a slight increase in the height and width of the J/ψ peak. Already these are indications that the neural network is somewhat biased towards the J/ψ mass, even on the simulated testing data.

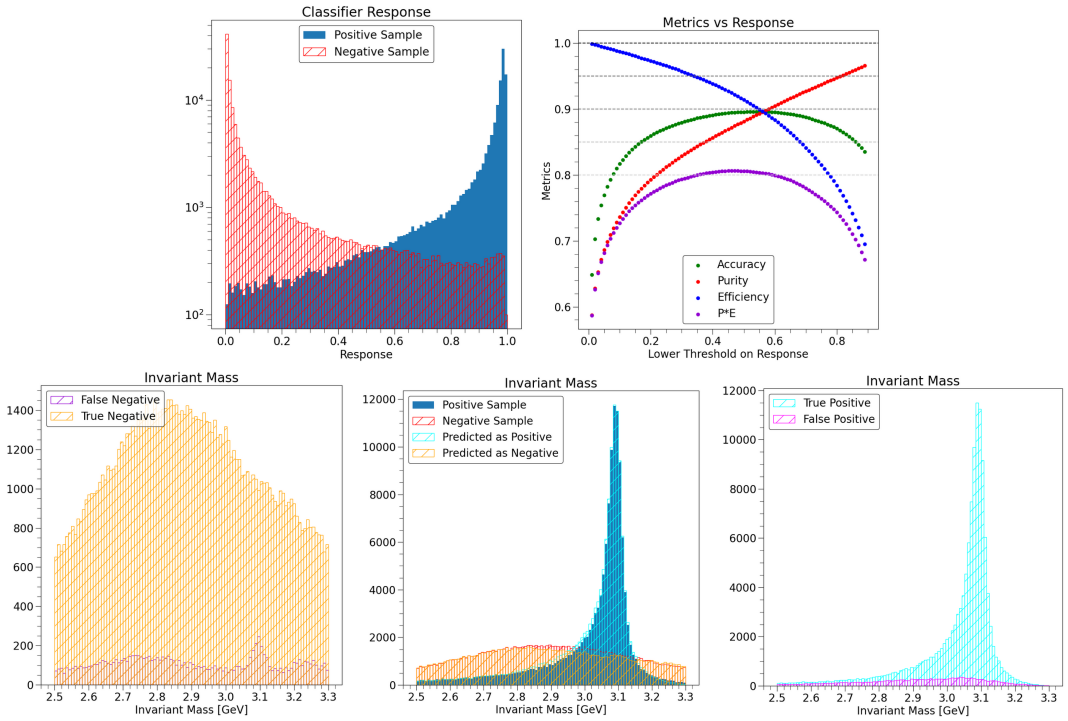


Figure 13: Top Row: The Response (Left) of the neural network classifier on the positive (blue) and negative (red) samples of the testing set, as well as metrics evaluated at various thresholds on the response (Right). Bottom Row: The invariant mass distribution of events rejected (left) and selected (right) by at cut at 0.5 on the response, and the invariant mass distribution of the predicted positives and negatives compared to the true positives and negatives (center).

The next step is to try out the neural network on data taken with CLAS12. Figure 14 shows the J/ψ and background yields as a function of the threshold on the response for J/ψ photoproduction data at CLAS12. A gaussian function was used to fit to the J/ψ peak, with a parameter of the fit returning the J/ψ yield, and a third order polynomial was fitted to the background. Although the background is significantly decreased, the J/ψ yield is increased for a response cut between 0.5 and 0.95. This is worrying as it is nonphysical; the neural network should not find additional J/ψ, only remove these and background. Similarly, the σ and mean of the gaussian fit is plotted at each threshold on the response in Figure 14. This shows how the σ increases as the threshold is increased, before decreasing once the threshold increases past 0.9. This is problematic as it suggests that the neural network is altering the fit hypothesis, which essentially betrays the fact that the neural network is biased towards the J/ψ mass.

Figure 15 shows the di-lepton invariant mass vs the network response. Although there are background events even at a response of 1, a vast majority of events close to the J/ψ mass have a response of 1. Figure 15 also

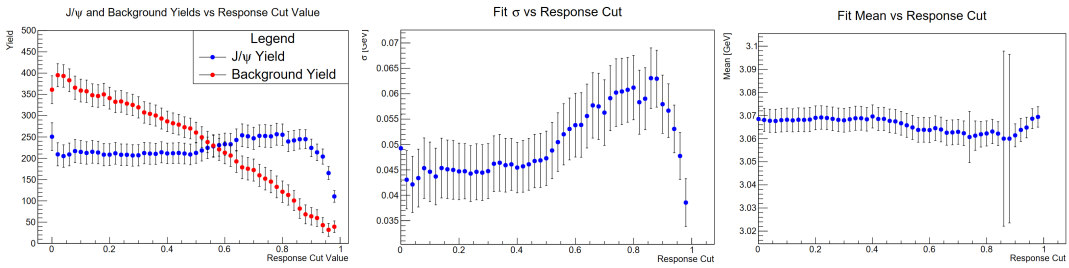


Figure 14: Top Left: The J/ψ and background yields as a function of the threshold on the response evaluated on CLAS12 data. Top Right: The σ of the gaussian fitted to the invariant mass distribution at each threshold on the response. Bottom: The mean of the gaussian fitted to the invariant mass distribution at each threshold on the response.

shows the di-lepton invariant mass spectrum at a threshold on the response of 0 (which is akin to not using the neural network), of 0.86 and of 0.94. Clearly at a response of 0.86 the width of the gaussian fitted to the J/ψ peak is increased which leads to an increase in the J/ψ yield. This is problematic when, for example, measuring cross sections as this would artificially increase the cross section. However, at a response of 0.94, the fit parameters are consistent with those returned without using the neural network. The background is also notably suppressed. A suggestion would therefore be to only use the neural network in conjunction with a high cut on the response.

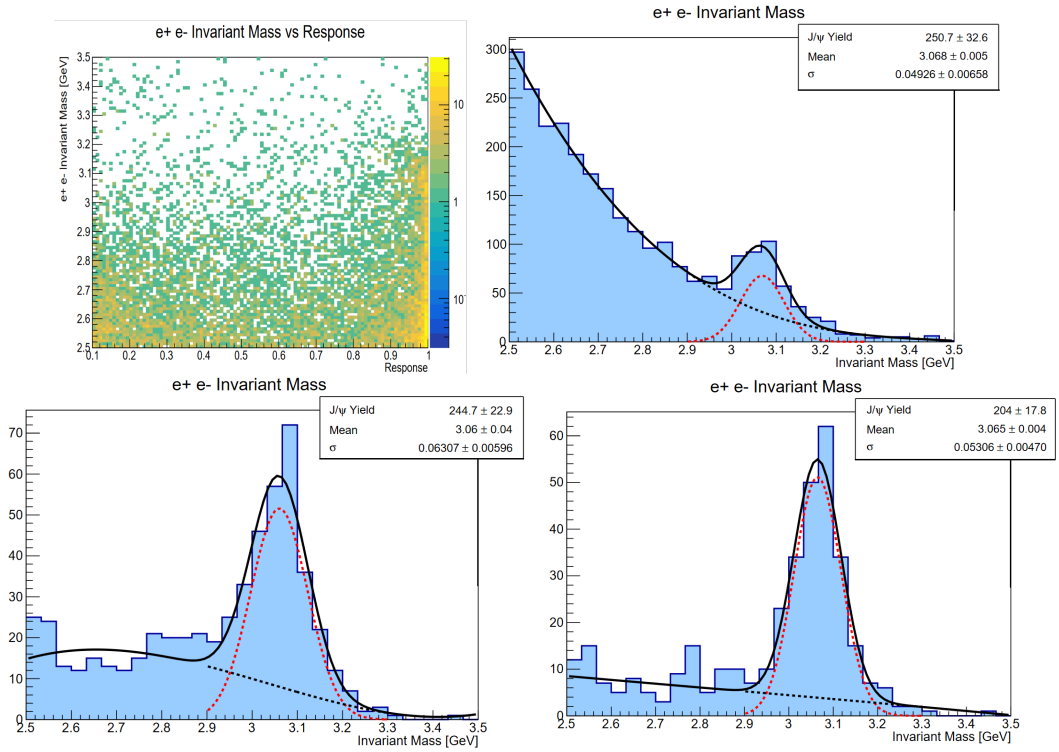


Figure 15: The di-lepton invariant mass distribution vs the network response and the invariant mass distributions at a threshold on the response of 0, 0.86 and 0.94. A threshold of 0 is akin to not using the neural network. A gaussian is fitted to the J/ψ peak with a third order polynomial fitted to the background.

Figure 16 shows a comparison of the J/ψ total cross section measured on the bound proton when using or not the machine learning based reaction identification tool. As shown, especially in the third bin, even a low cut on the response of 0.15 has a significant impact on the total cross section measurement. The largest divergence occurs at a response cut of 0.86 which corroborates the conclusions established from Figures 14 and 15. Overall

it should be rather clear by now that these types of tools are unsuited to cross section measurements as they introduce a bias that can increase the total cross section.

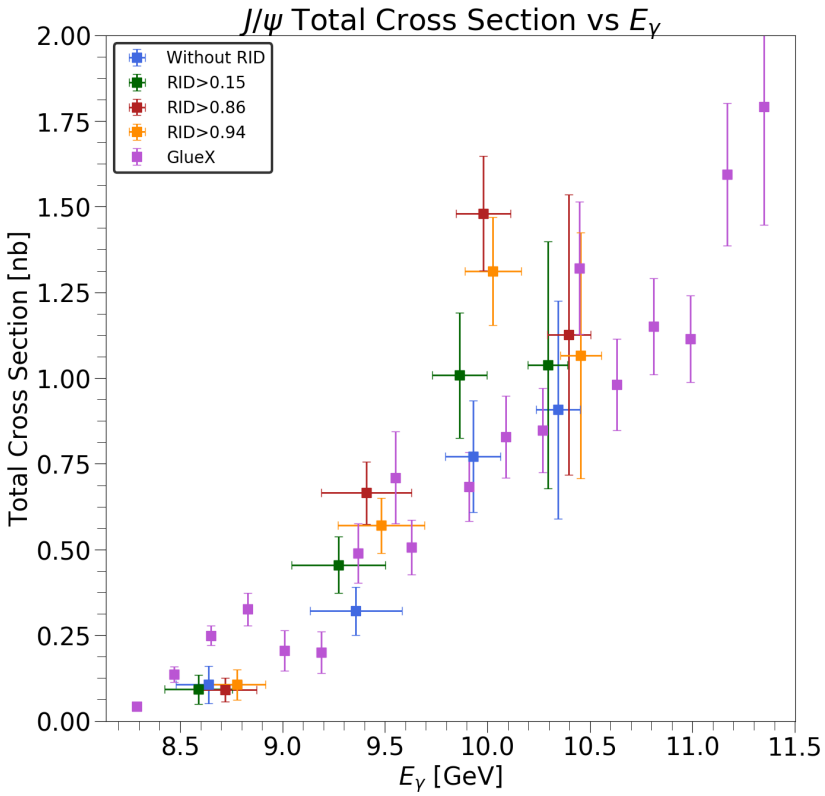


Figure 16: The J/ψ total cross section, measured in the $ep_{bound} \rightarrow (e')e^+e^-p$ channel with the machine learning based reaction ID and compared to the GlueX results (magenta). The cut on the response is set at 0.15 (green), 0.86 (red) and 0.94 (orange), or the reaction ID tool is not used (blue).

Overall, great care should be taken when using machine learning based methods like the one described in this section. Figures 13, 14, 15 and 16 demonstrate clear evidence of a bias introduced in the analysis by this method, which directly impacts the extraction of physics results such as the cross section. However, as shown in Figures 14 and 15, the background is greatly suppressed by the neural network. The network also has a high prediction rate of order $10^5 s^{-1}$. Reaction identification tools such as the one developed here can still be useful for different applications. For example, the reaction identification tools may be useful for measurements relying on asymmetries where the difference of two observables is divided by their sum, as any increase in the observables would factor out. The bias introduced here is also mainly problematic due to the reliance on fits to the invariant mass to extract the J/ψ yield. With detailed and accurate simulations of both signal and background, template fits could be used that would account for the network bias to the J/ψ mass without biasing the extraction of the J/ψ yield (although this would also remove the need to do the reaction identification in the first instance).

We can assess the analysis by filling out the checklist defined in Table 4. Note that the list of plots was not available at the time when the analysis was made. Firstly, AI/ML tools are not necessary as a cut based analysis with cuts on the missing mass squared and Q^2 is sufficient. The training data is made using simulated data with the simulated signal containing a J/ψ peak. Correlations between the input variables (final state particles four-momenta) and the target observable, the J/ψ cross section, are obvious. The comparison to conventional methods demonstrates a bias, as does varying the threshold on the algorithm output when checking the impact on kinematics, observables and fits. Overall, the checklist demonstrates the pitfalls of this analysis which would lead to biases being introduced into measurements of the J/ψ cross section.

C.2.2 Review checklist

Review item	Questions	Validation
Task Selection	Does the task really require AI/ML ? Can a simpler AI/ML task be used instead ?	No Cut based analysis sufficient
Training Data	Is it representative of the data the algorithm will be applied to? Is the use (or not) of simulated data justified? If simulated, does it have physics in it? If not-simulated, does it avoid contamination?	Needs to use simulation. Training data representative of application data. Training data has J/ψ signal.
Feature Selection	Are the input features well motivated ? Was the feature importance evaluated?	Yes No
Algorithm and Metric Definition	Can the algorithm and metrics be understood solely by reading the note ?	Yes Yes
Hyperparameter Selection	Are the algorithm and its hyperparameters explicitly mentioned and justified ?	Mentioned, not justified
Plots	Are the suggested plots present?	Some
Correlations to Observables	Are the correlations between algorithm output and observables evaluated? If needed, were de-correlation procedures applied?	No but obviously present. No
Validation Sets	Were metrics produced on non-simulated data? Was the algorithm applied to synthetic data? Were input output tests done?	Yes No No
Comparison to Conventional Method	Is the comparison possible and present?	Yes
Threshold Checks	Is the threshold on the algorithm output varied to evaluate the impact on kinematics, observables and fits?	Yes

D Deploying AI/ML Tools

There are two main CLAS12 software packages that are suited to share/deploy AI/ML based tools at CLAS12 (ignoring for now possible online implementations of AI/ML algorithms). Coatjava [32] is the main software package responsible for the offline reconstruction of CLAS12 data, dealing with a variety of steps ranging from the formation of clusters and tracks from hits in CLAS12 detector subsystems, to parsing information from different subsystems into single particles along with assigning them a particle ID [33]. The Implementation Guardian of Analysis Algorithms (Iguana) [34] is aimed at preserving and centralising common analysis algorithms, such as fiducial cuts, momentum corrections, and kinematics reconstruction. Iguana is not a framework for reading data, rather it is a set of algorithms that produces additional data (eg kinematics reconstruction) or mutates the data (eg momentum reconstruction).

How AI/ML based tools are deployed at CLAS12 should depend on three aspects that quite clearly map onto either Coatjava or Iguana.:

- Purpose: Are these tools aimed at improving the reconstruction, analysis, data taking (...)?
- Impact: Will these tools require the data to be recooked or recalibrated?
- Input Data: Are these tools using information available in standard DSTs?

Tools deployed in Coatjava:

- Are aimed at the data reconstruction.
- Feed into reconstruction steps and may affect data calibration (eg track finding).
- Use non standard DST info (eg adc/tdc banks).

Tools deployed in Iguana:

- Are aimed at analysis post reconstruction and don't feed into reconstruction.
- May need to be re-run by analyzers without needing reconstruction/recalibration of the dataset.
- Use standard DSTs (eg REC banks).

Particle Identification (PID) is a good example of a task that might be confusing with regards to how it should be deployed. The event builder PID is done during the data cooking process, whilst PID improvements algorithms are often implemented at the analysis level. By following the three points above, one can easily decide whether a PID algorithm should be deployed in Coatjava or Iguana. Efforts have gone into online electron identification and the relevant tools obviously would be deployed online. In general however, PID does not aim to improve the reconstruction and quality of data but rather to improve the data analysis and production of physics results. Similarly, PID algorithms generally will not require the data to be recooked or recalibrated. The main question here is what the input data to the algorithm are. The standard DSTs often contain most of the data that is required to identify final state particles. If these data are sufficient for a PID algorithm, then it should be deployed in Iguana. There may be some cases where the standard DSTs do not contain sufficient information, and non-standard DSTs are required. In this case, the algorithm should be deployed in Coatjava.

Based on discussions within the collaboration, a couple of points should be emphasized. Firstly, algorithms in both Coatjava and Iguana have to be thoroughly validated, trustworthiness should not be associated to where the algorithm is available. Secondly, if demand justified it, Iguana could be adapted such that it can be run as an extra CLARA service post-reconstruction, for example in the case where there is demand to have certain Iguana algorithms applied to a given run group dataset. As such, algorithms should not be put in Coatjava just to avoid having to run Iguana or to enhance their availability.

At the time of writing, the process to review common AI/ML analysis tools has not been decided upon by the collaboration. We suggest two ways of doing so:

- The first analysis note that includes a given tool has to validate it. Subsequent analyses can quote the review.
- Any AI/ML tool added to Coatjava/Iguana has to be reviewed independently. This will require writing a note and going through a review process which is then quoted by any analysis note that includes the tool.

E Results of the initial survey on ML-based analysis reviews

This section will be removed from the final guidelines.

This section summarizes the answer received in the initial survey which led to the draft of these guidelines.

E.1 Questions

The survey asked several questions, recapped below:

Question 1: Please describe the AI/ML tool or analysis you were asked to review:

Question 2: Did you find this review particularly challenging compared to others without AI/ML components?

Question 3: Please give an overview of the questions and further studies you asked for when reviewing the AI/ML component of the analysis:

Question 4: Did you have trouble with understanding:

- The mechanics of the algorithm and technical terms:
- Figures or metrics presented by the analyser:
- The impact of the method on the results:
- How to validate the method:
- What to look for and what might be missing in the analysis:

Question 5: Please feel free to comment further:

E.2 Answers

All answers are copied below (more or less verbatim). We will differentiate between answers received before and after these guidelines have been shared with the collaboration, and before and after edits to the guidelines. We might change how this is presented if the number of answers grows such that this is unmanageable. The first question is omitted to avoid drawing comparisons between the answers and specific analyses.

Question 2: Did you find this review particularly challenging compared to others without AI/ML components?

- Somewhat yes, for 4 reasons at least:
  - When using ML technics, the analysis author sometimes omits to describe the 'classical' part of their analysis, which induces delays
  - ML is sometimes used to hide problems, and rarely compared with 'classical analysis' when relevant
  - The validation of the ML technics/algorithm is sometimes neglected (see previous point)
  - ML technics are sometimes used in region where they have not been trained, and a specific validation for this should be done
- I'm not an expert in AI/ML, therefore I'm not able to judge the details of the various techniques implemented. I mostly focused on reviewing how they are implemented and applied to get the results.
- If the ML method is the main aspect of the study, other focuses have to be set for the review than for a classical analysis with simple cuts. Especially more details and plots are needed to really verify the validity of the procedure.
- Yes, somewhat. The challenge stemmed not from the BDT method itself, but from the way it was applied in a kinematic region not covered by existing data, raising concerns about extrapolation and validation.

Question 3: Please give an overview of the questions and further studies you asked for when reviewing the AI/ML component of the analysis:

- Reviewer:
  - Better understand the training samples: when MC is used to train, the training samples should reproduce the dataset on which it is used in a reasonable manner (especially if it is used for event selection). For this specific analysis, we asked to retrain completely the model because the MC would not describe the data (at the physics level, ie some underlying processes were not taken into account in the description of the background)
  - Understand the response of the method: validate on both full MC and controlled data sample when possible
  - Verify the correlation of the relevant kinematic variable of the analysis with the response of the algorithm for MC and controlled data sample (especially important for event selection)
  - Justify the relevance of the approach: when possible, compare with other/simpler approach (for example in BDT one can quantify the separation power of the input variables, and thus one can compare to a simple cut-based analysis on these variables). Doing this also helps with the validation.
  - Further validation using MC if the method is used in a region where it was not train (ie event selection with input variables outside the range of training)
  - Understand/control the systematic errors of the ML method
- Reviewer:
  - Choice of the variables for the training of the AI/ML model. Sometimes the choice does not appear fully justified. Why are some kinematic variables used and others discarded? Is there a well defined quantitative way to choose? Or is it just a matter of personal taste?
  - Training of the model. Typically, the AI/ML models are trained on Monte Carlo data and then applied to real data. But is the agreement between Monte Carlo and real data sufficiently good for that? What is the impact of e.g. different momentum, time, etc. resolutions between real and simulated data?
  - Comparison with traditional analysis. We have published many papers using traditional analysis procedures, but AI/ML techniques are becoming more and more used and will be the standard at some point. Therefore, the comparison between the two methods should be carefully investigated and understood.
- Reviewer:
  - More plots of the input variables and especially a more detailed data – MC comparison.
  - A special focus needs to be set on the reproduction of the data by the MC
  - Multidimensional comparisons were requested.
- I requested a more robust justification for using the BDT in a region not covered by data from the corresponding traditional analysis. Specifically, I asked for quantitative tests of the model's behavior in this extrapolated region (via simulations/synthetic data) and suggested studying its stability under different training conditions. I also requested clearer metrics demonstrating the performance and reliability of the BDT outside the training domain.

Question 4: Did you have trouble with understanding:
Question 4a: The mechanics of the algorithm and technical terms:

- No.
- The mechanics of the algorithm is almost a black box for me; the technical terms are just something one has to get used to, provided they are clearly explained.
- No.
- No.

Question 4b: Figures or metrics presented by the analyser:

- No.
- Same as before, the analyzers have to clearly explain how they are defined and their meaning.
- No.
- No, though additional metrics would have been helpful.

Question 4c: The impact of the method on the results:

- No.
- See discussion with comparison with traditional methods.
- It needs to be validated, if the ML method produces correct results or fake results (i.e. with a modulation but not a rejection of backgrounds) which only look correct.
- Yes, due to insufficient validation in the extrapolated region.

Question 4d: How to validate the method:

- No.
- This is really the heart of the question. See discussion on MC comparison above
- We requested additional validation checks, like implementing modulation in the data and the background MC and trying to reconstruct them.
- Validation was a concern in this case. I emphasized the need for tests that evaluate the robustness and generalization of the BDT in regions lacking training data, such as cross-checks using synthetic data, bootstrapping, or comparison with baseline cut-based selections.

Question 4e: What to look for and what might be missing in the analysis:

- No.
- Sometimes the results of the AI/ML algorithm could be counter intuitive. In that case one should make all the possible efforts to understand why it is so. It is more work, but I believe it is the only way to really learn something new from AI/ML.
- No response.
- I looked for performance benchmarks, overtraining checks, uncertainty quantification, and comparisons with physically motivated baselines. The primary missing component was a study of how reliable the BDT predictions are in regions not covered by the traditional analysis.

Question 5: Please feel free to comment further:

- These comments are mostly relevant for a classification use of ML (which is the majority of the use cases in the collaboration so far).
- I have the feeling that people tend to grant more credit to results obtained using sophisticated software like AI/ML. While it is certainly true that AI/ML techniques allow to exploit the physics information in the experimental data at best, their output heavily relies on the way the models are trained. A wrong training of a perfect AI/ML model will lead to wrong result.
- No response.
- While machine learning tools like BDTs can be powerful, their use in extrapolated or poorly constrained kinematic regions must be accompanied by careful validation and uncertainty estimation. This is especially critical when the physics conclusions depend on regions where data-driven training is not possible.

E.3 Other Feedback:

We also welcome any other feedback from reviewers of AI/ML analyses, even if it doesn't follow the format laid out in the questions above.

Reviewer: I do not think the ML methods likely to be used should be more of an issue for reviewers than things like sPlots or partial waves analysis.
The real question is are the analysers using the correct tool, or does it bias their results. For example I would be very critical of using solely kinematics in a ML algo for event selection. There is also the question of training data, and have the analysers used an appropriate training set.
In general any algorithm should be properly validated, so how do you enforce good validation procedures? The review committee should not need to understand the underlying details, but should be made aware of what goes in and what comes out and what choices the analyser has made in tuning it. I think having a sepereate analysis not on a specific algorithm could help. This could then be reviewed by a competent reviewer, thus taking part of the burden from the review committee.

References

[1] D.I. Glazier, R. Tyson, Converting sWeights to Probabilities with Density Ratios, *ArXiv e-prints* arXiv:2409.08183v2 (2024).

[2] J. Brownlee, How to Calculate Feature Importance With Python, Machine Learning Mastery (2020). https://machinelearningmastery.com/calculate-feature-importance-with-python/.

[3] L. Bradshaw, R. K. Mishra, A. Mitridate, B. Ostdiek, Mass Agnostic Jet Taggers, *ArXiv e-prints* arXiv:1908.08959v2 (2019).

[4] A. Burkov, *The Hundred-Page Machine Learning Book* 2019 Open Access at http://themlbook.com/wiki/doku.php .

[5] V. Pleskot, D. Scheirich, Statistical Methods (2020). YouTube channel https://www.youtube.com/@statisticalmethods6041 .

[6] G. Sanderson, 3Blue1Brown (2015). YouTube channel https://www.youtube.com/c/3blue1brown .

[7] M. T. Ribeiro, S. Singh, C. Guestrin, "Why Should I Trust You?" Explaining the Predictions of Any Classifier, *ArXiv e-prints* arXiv:1602.04938v3 (2016).

[8] D.I. Glazier, R. Tyson, Converting sWeights to Probabilities with Density Ratios, *ArXiv e-prints* arXiv:2409.08183 (2024).

[9] R. Tyson, G. Gavalian, D. Ireland, B. McKinnon, Deep Learning Level-3 Electron Trigger for CLAS12, *Comp. Phys. Comm.* **290**, 108783 (2023).

[10] M.Cavaioni, Machine Learning: Decision Tree Classifier, *Medium* (2017). https://medium.com/machine-learning-bites/machine-learning-decision-tree-classifier-

[11] J. Furnkranz, *Encyclopedia of Machine Learning*, Springer US, 2010, pp 263–267.

[12] S. K. Murthy, Automatic construction of decision trees from data: A multi-disciplinary survey, *Data Mining and Knowledge Discovery* **2** 4 (1998).

[13] Y. Freund, R. E. Schapire, Experiments with a new boosting algorithm, *Proceedings of the 13th international conference on machine learning* 148–156 (1996).

[14] J.H. Friedman, Greedy function approximation: a gradient boosting machine, *Annals of Statistics* **29** 5 (2001).

[15] A. Connolly (AstroML developers), Deep Learning: Classifying Astronomical Images, *AstroML Interactive Book* (2022). https://www.astroml.org/astroML-notebooks/chapter9/astroml_chapter9_Deep_Learning_Classifying_Astronomical_Images.html.

[16] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press, 2016.

[17] D. P. Kingma and J. Ba, Adam, Adam: A method for stochastic optimization, *ArXiv e-prints* arXiv:1412.6980 (2014).

[18] F. Murtagh, Multilayer perceptrons for classification and regression, *Neurocomputing*, **2** 5 (1991).

[19] S. Albawi, T. A. Mohammed and S. Al-Zawi, Understanding of a convolutional neural network, *2017 International Conference on Engineering and Technology (ICET)*, doi: 10.1109/ICEngTechnol.2017.8308186, (2017).

[20] K. O'Shea, and R. Nash, An Introduction to Convolutional Neural Networks, *ArXiv e-prints* arXiv:1511.08458 (2015).

[21] D. Bank, N. Koenigstein, R. Giryes, Autoencoders, *ArXiv e-prints* arXiv:2003.05991 (2020).

[22] M. Abadi, *et al.*, TensorFlow: Large-scale machine learning on heterogeneous systems, *ArXiv e-prints* arXiv:1603.04467v2 (2016). Software available from tensorflow.org.

[23] A. Paszke, *et al.*,PyTorch: An Imperative Style, High-Performance Deep Learning Library, *ArXiv e-prints* arXiv:1912.01703 (2019). Software available from pytorch.org.

[24] F. Pedregosa, *et. al.*, Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research* **12** 2825 (2011).

[25] Deep Java Library (DJL): https://github.com/deepjavalibrary/djl/tree/master

[26] E. Krasanakis, S. Papadopoulos, I. Kompatsiaris, JGNN: Graph Neural Networks on native Java, *SoftwareX* **23** 101459 (2023).
https://github.com/MKLab-ITI/JGNN

[27] J. Therhaag, TMVA Toolkit for multivariate data analysis in ROOT, *PoS* **ICHEP2010** 510 (2010).

[28] K. Albertsson, *et. al.*, ROOT TMVA user guide.
https://root.cern.ch/download/doc/tmva/TMVAUsersGuide.pdf.

[29] M. Tenorio-Pita, Enhancing Lepton Identification in CLAS12 using Machine Learning Techniques, CLAS12 Note 2024-005.
https://misportal.jlab.org/mis/physics/clas12/viewFile.cfm/2024-005.pdf?documentId=172.

[30] Y.G. Sharabian, *et al.*, The CLAS12 high threshold Cherenkov counter, *Nucl. Inst. Methods A* **968** 163824 (2020).

[31] R. Tyson, J/ψ Near-Threshold Photoproduction off the Proton and Neutron with CLAS12, Ph.D. Thesis, University of Glasgow (2023).
https://theses.gla.ac.uk/83777/ .

[32] coatjava: https://github.com/JeffersonLab/coatjava

[33] V. Ziegler *et al.*, The CLAS12 software framework and event reconstruction, *Nucl. Instrum. Methods A* **959** 163472 (2020).

[34] Iguana: https://github.com/JeffersonLab/iguana/tree/main