Machine Learning Approaches to Final State $\pi^{\pm}$ Identification in ProtoDUNE SP

William Fahie

September 2025

1 Introduction

Final state charged pion identification is a central task in ProtoDUNE Single-Phase (PDSP) analyses. Traditional cut-based selections, while robust, achieve limited efficiency and purity. Machine learning (ML) methods offer an opportunity to exploit correlations across multiple observables, providing superior classification performance. This report summarises results from applying decision trees, random forests, and gradient-boosted trees to $\pi^{\pm}$ classification, with a comparison to the cut-based approach from Bhuller [1] to quantify the significant gains in $\pi^{\pm}$ selection efficiency and purity.

2 Background

Neutrinos are one of the most important areas of research in modern physics. Understanding their properties, namely neutrino oscillations, may give insight into some of the universe's most fundamental questions, like why matter dominates over anti-matter. Being neutral and almost massless, neutrinos only interact via the weak interaction, making them very difficult to measure directly. Instead, we measure the products of their interactions with matter and work backwards.

2.1 DUNE

The Deep Underground Neutrino Experiment (DUNE) is a long-baseline neutrino experiment hosted by Fermilab in the USA. It consists of a neutrino beam, a Near Detector (ND) at Fermilab, and a Far Detector (FD) at Stanford's Underground Research Facility, as illustrated in Figure 1. The FD's role is to detect the neutrinos that complete the 1300 km journey from the source. This is a huge statistical challenge, with the FD's modules being the largest Liquid Argon Time Projection Chambers (LArTPCs) ever built.

When high-energy neutrinos enter one of DUNE's LArTPC modules, a small fraction of them will strike a proton or neutron inside an argon nucleus - known as the primary interaction. This can often create new particles, including pions. Before leaving the nucleus, a newly created pion can undergo further interactions with other nucleons, known as Final State Interactions (FSI).

The produced final-state particles, which appear to emerge from the primary interaction vertex, travel through the liquid argon, leaving a trail of ionisation. The properties ("observables") of these trails can be reconstructed by the TPC's detectors to produce Particle Flow Objects (PFOs). By classifying the PFOs using their observables, we can work backwards to understand the nature of the FSIs and consequently measure neutrino fluxes.

Pion FSIs can be particularly complex, including charge exchange, where a charged pion transforms into a neutral pion. These FSIs can present significant uncertainties when measuring neutrino fluxes. Thus, in recent years, an increased effort has been made to study pion-argon interactions.

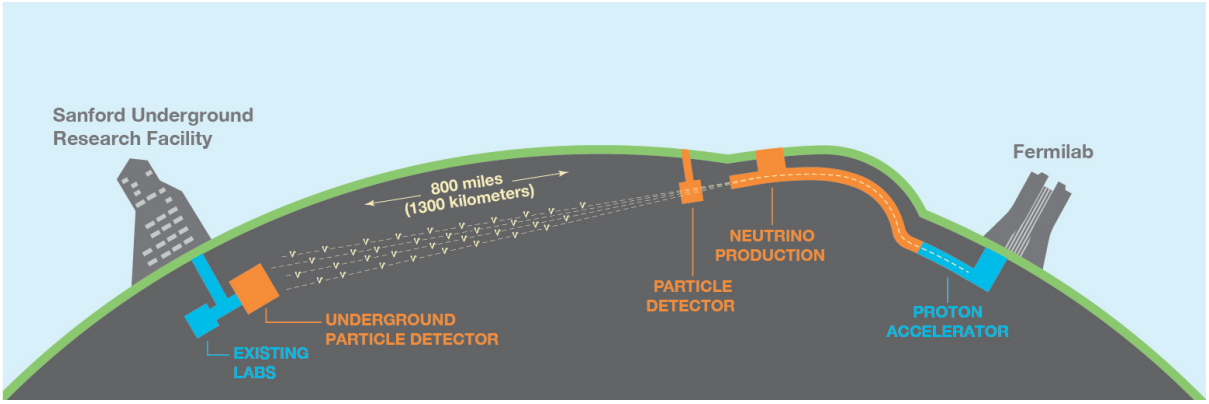

Figure 1: Cartoon of the DUNE experiment, showing the neutrino beam, near detector, and far detector. Credit: Fermilab.

2.2 ProtoDUNE

ProtoDUNE is an experiment at CERN's neutrino platform whose goal is to prototype the production and installation procedures for DUNE's FD designs. In 2018, PDSP successfully collected large samples of test beam data for various momenta between 0.3 GeV and 7 GeV. Of importance to us is the pion test beam data, as this allows us to explore and better understand the aforementioned pion-argon FSIs and produce calibrations for the DUNE FD designs. In particular, our study focusses on 2 GeV $\pi^{\pm}$ data.

By counting the number of pions emerging from an interaction vertex, one can determine the nature of a particular FSI. So the goal then becomes to perform the most accurate classification of final-state pions possible. This report demonstrates a machine learning approach to the classification of final-state pions using Monte-Carlo (MC) simulation data that replicates the PDSP.

3 Cut-based approach

The method used to classify final-state pions in Bhuller's thesis is a cut-based approach. The benefit of using MC data to reconstruct the PFOs is that we have access to their true Particle Identification (PID). Consequently, one can find certain thresholds that yield a subset of data with high pion purity and efficiency. Following from an initial beam selection (a prior cut-based approach used to ensure a high π^{+} purity in the beam particles), Bhuller's cuts and their final performance are shown in Table 1. Purity and efficiency are defined in Equations 1 and 2, where the signal is PFOs with the true PID "$\pi^{\pm}$".

Cut	Count	Purity (%)	Efficiency (%)
Beam particle selection	62877	19.1 ± 0.2	100
$(\chi^2/ndf)_p > 61.2$	58237	25.8 ± 0.3	92.6 ± 0.1
Track length > 27.1 cm	40452	34.6 ± 0.5	64.3 ± 0.2
Track score > 0.5	38881	47.3 ± 0.6	61.8 ± 0.3
1.6 < Median dE/dX < 2.8 MeV/cm	33710	51.0 ± 0.9	53.6 ± 0.4

Table 1: Performance of the cut-based $\pi^{\pm}$ selection in the MC, as presented in Bhuller's thesis [1].

So the baseline performance for final-state $\pi^{\pm}$ selection is a purity of 51.0% and an efficiency of 53.6%, which leaves a lot of room for improvement. A significant drawback of the cut-based method is that each cut is applied independently, failing to account for the complex correlations

between observables. The particular observables used in Table 1, and many more that we use in our approach, are explored in Section 5.

$$\text{purity} = \frac{\text{number of signal PFOs selected}}{\text{total number of PFOs selected}} \tag{1}$$

$$\text{efficiency} = \frac{\text{number of signal PFOs selected}}{\text{total number of signal PFOs after beam selection}} \tag{2}$$

4 Machine Learning Classifiers

The objective of this project was to optimise the final-state pion selection using ML models, where the PFO's observables are the *features* and the PIDs are the *class labels*. Throughout the project, we train the models in three ways:

1. **Full Classification:** The model attempts to classify particles as any of the possible PIDs in the training data. The pion performance is then taken from this.
2. **Binary Classification:** The model only attempts to classify between pions (1) and not-pions (0).
3. **Balanced Binary Classification:** The same as Binary Classification, but we balance the classes beforehand by randomly dropping non-pion PFOs from the data until the pion and non-pion classes are approximately balanced.

Given the relatively small number of observables and the scale of this classification, tree-based models were the natural architecture to use. While neural networks can achieve impressive results, they are far more complex to engineer and are typically more useful for higher-complexity ML tasks like image recognition or procedural text generation.

4.1 Decision Tree

The most fundamental machine learning architecture is the decision tree [2]. In essence, this is a more extensive version of the cut-based approach. A tree consists of nodes where, at each parent node, a cut is made on the data (e.g. track length > 20 cm) and the two resulting subsets (true and false) branch to different child nodes. The final nodes of a decision tree are known as leaf nodes. The majority class of the training data within a leaf node becomes the label for that leaf. When a new sample is introduced, it traverses down the tree following the conditions at each node, and when it reaches a leaf, that leaf's label becomes the DT's classification of that sample.

A standard Decision Tree (DT) learns by iteratively testing different cut thresholds to find the ones that optimise a particular metric. The classic metric is Gini impurity, defined in Equation 3.

$$G = \sum_{i=1}^{C} p_i(1 - p_i) = 1 - \sum_{i=1}^{C} p_i^2 \tag{3}$$

In Equation 3, C is the total number of classes, and p_i is the probability of a randomly chosen sample from the node belonging to class i. A Gini impurity of 0 would represent a "pure" node, where all samples belong to a single class.

The tree-building algorithm iteratively searches for the feature and cut value that results in the greatest "information gain" - equivalent to the largest decrease in the weighted average Gini impurity of the resulting child nodes. This process is repeated recursively until a stopping

condition is met. Figure 2 shows a basic DT with a depth of 2, trained on a subsample of the MC data.

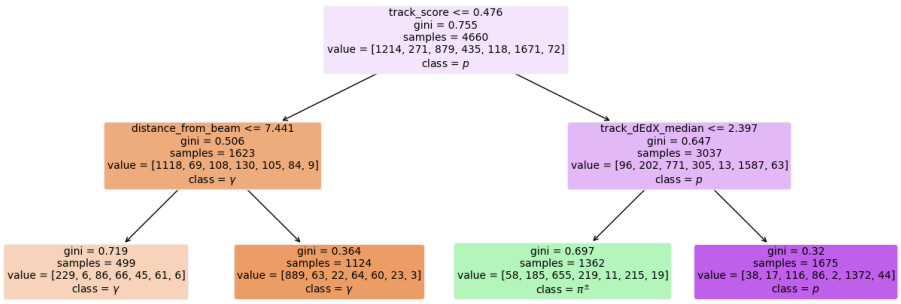

Figure 2: A decision tree classifier with a maximum depth of two, trained on 4660 PFOs. Each node displays the splitting condition, the Gini impurity (Eq. 3), the total number of samples, and the majority class. The `value` array lists the number of samples for each class in the order: [γ, $\mu^{\pm}$, $\pi^{\pm}$, $\pi^{\pm}$:2nd, e^{+}, p, other].

While highly interpretable, a single decision tree is prone to **overfitting**. Since the fitting algorithm is "greedy," at each node it picks the single best feature and cut value to reduce impurity *at that moment*, without looking ahead to see how that split might affect the overall tree structure. In the most extreme case, it could create a unique path and a final leaf node for every sample.

An overfitted model "memorises" its training data instead of learning its patterns and consequently fails to generalise to unseen data. We can attempt to avoid overfitting by specifying properties of the tree ("hyperparameters") such as a maximum depth or a minimum number of samples a node must have to split. The downside is that this limits the model's complexity, likely lowering its general performance. To account for overfitting while retaining complexity, we turn to ensemble methods that combine the predictions of multiple trees.

4.2 Random Forest

A Random Forest is an ensemble learning method [3] that uses a collection of decision trees and outputs the modal class of the individual classifications. It introduces two key concepts to ensure the trees are decorrelated from one another:

1. **Bagging (bootstrap aggregating):** Instead of training every tree on the entire dataset, each tree is trained on a random subsample of the data drawn *with replacement*. This means some data points may be used multiple times in a single tree's training set, while others may not be used at all.

2. **Feature Randomness:** When splitting a node, each decision tree is not allowed to search over all available features. Instead, it is restricted to a random subset. This prevents a few dominant features from controlling the structure of all the trees, thereby promoting diversity among them.

By averaging the predictions of many diverse, decorrelated trees, the Random Forest reduces variance and produces a model that generalises much better to new data than a single, complex tree. For implementing these models, we use the **Scikit-Learn** library, which provides optimised and user-friendly implementations.

4.3 Gradient Boosted Decision Trees

Gradient Boosted Decision Trees [4] are another powerful ensemble technique, but they operate on the principle of **boosting** rather than bagging. Unlike Random Forests, where trees are built independently, gradient boosting builds trees sequentially, where each new tree attempts to correct the errors made by the previous ones.

The process can be summarised as follows:

1. An initial, simple model (e.g., a shallow tree) is trained.
2. The gradients of a loss function are calculated with respect to the predictions. These represent the errors, or residuals, between the predicted and true values.
3. A new tree is trained, not on the original labels, but on these residuals. Its goal is to learn the *mistakes* of the current ensemble.
4. The predictions of this new tree are added to the ensemble's predictions, scaled by a small **learning rate** to prevent overfitting.
5. This process is repeated, with each new tree incrementally improving the overall model by focusing on the remaining errors.

For this work, we use the **XGBoost** library [5], a highly optimised implementation of gradient boosting. The standard loss function is logistic loss. For binary classification, this is:

$$L(y, \hat{p}) = -[y \log(\hat{p}) + (1 - y) \log(1 - \hat{p})] \tag{4}$$

where y is the true label (0 or 1) and $\hat{p}$ is the predicted probability. For full classification with K classes, it generalises to:

$$L(y, \hat{p}) = -\sum_{k=1}^{K} y_k \log(\hat{p}_k) \tag{5}$$

where y_k is a binary indicator if the sample belongs to class k.

5 Data and Observables

The full MC dataset contains 665,440 events, each containing multiple reconstructed PFOs. After applying the same beam selection cuts as in Bhuller's thesis and dropping PFOs with missing data, we are left with 367,790 PFOs. The data is then randomly split into a training dataset (80%) and a testing dataset (20%). These PFOs include the signal particles ($\pi^{\pm}$) and background particles from interactions and cosmic sources. For each PFO, a set of 21 observables is extracted.

5.1 Calorimetric Observables

- `track_dEdX_median` and `track_dEdX_mean`: Median and mean energy deposited per unit of track length (MeV/cm).
- `track_dQdX_median` and `track_dQdX_mean`: Median and mean charge collected per unit of track length.
- `dEdX_end_pos`: The mean dE/dX near the end of the track.

5.2 Geometric Observables

- `track_length`: Total reconstructed track length in cm.
- `track_start_pos_[x,y,z]` and `track_end_pos_[x,y,z]`: 3D coordinates of the track's start and end points.
- `angle_to_beam`: Angle between the PFO's trajectory and the initial beam particle.
- `angle_to_vertical`: Angle of the track with respect to the vertical axis.
- `residual_range_mean` and `residual_range_median`: The extra distance a particle travels over its range calculated from its energy deposition.

5.3 Track Observables

- `track_chi2/ndof_[proton, pion, muon]`: The χ^2 per degree of freedom from fitting the PFO's dE/dX profile to expected energy loss curves for the proton, pion and muon respectively.
- `track_score`: A CNN-based score from 0 to 1 indicating how "track-like" a PFO is.
- `track_vertex_michel`: A score indicating the presence of a Michel electron near the end of the track.
- `track_vertex_nhits`: The number of hits associated with the PFO's vertex.

5.4 Derived Observables

- `hit_density`: The number of hits on the collection plane per unit track length.
- `dEdX/dQdX`: The ratio of the mean calibrated energy loss to the mean raw charge deposited.
- `dEdX_per_length`: The mean dE/dX normalised by its total length.

The relationships between these observables can be seen in Figure 3. Highly correlated observables offer negligible gains in model performance, so for computational efficiency, we removed observables with a correlation $> 75\%$. The 1D distributions for the observables are shown in A.

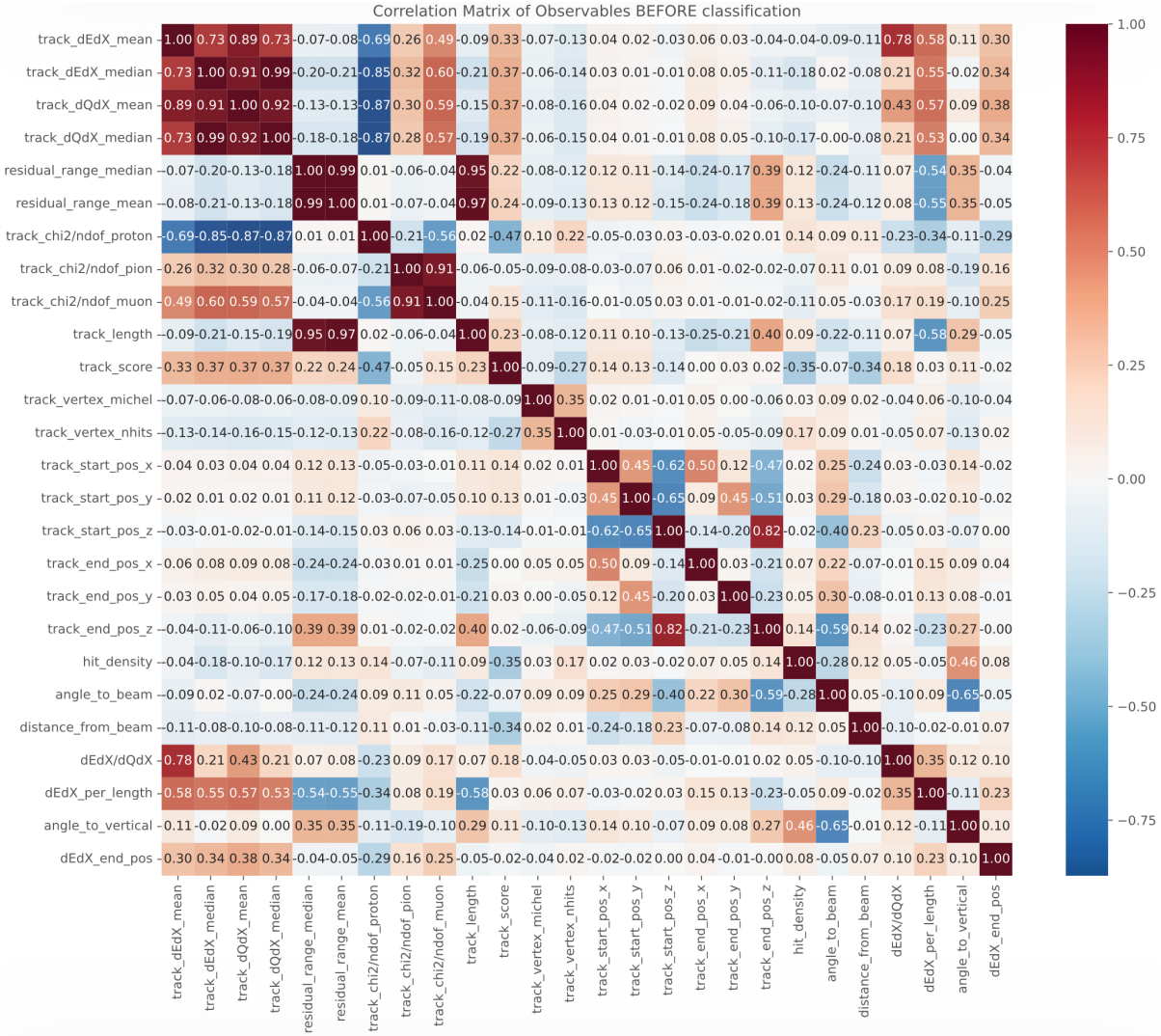

Figure 3: Correlation matrix showing the linear relationships between the PFO observables in the full MC dataset. Strong correlations (dark red or dark blue) suggest redundant information.

6 Results

6.1 Decision Tree

A key hyperparameter for a decision tree is its `max_depth`. Figure 4 shows how purity and efficiency scores change with maximum depth. We plot both train and test performances to identify where they diverge (overfitting). We see this begins around `max_depth=8`.

Using this stopping condition, Tables 2, 3, and 4 show the performances for the DT trained for full, binary, and balanced binary classification, respectively.

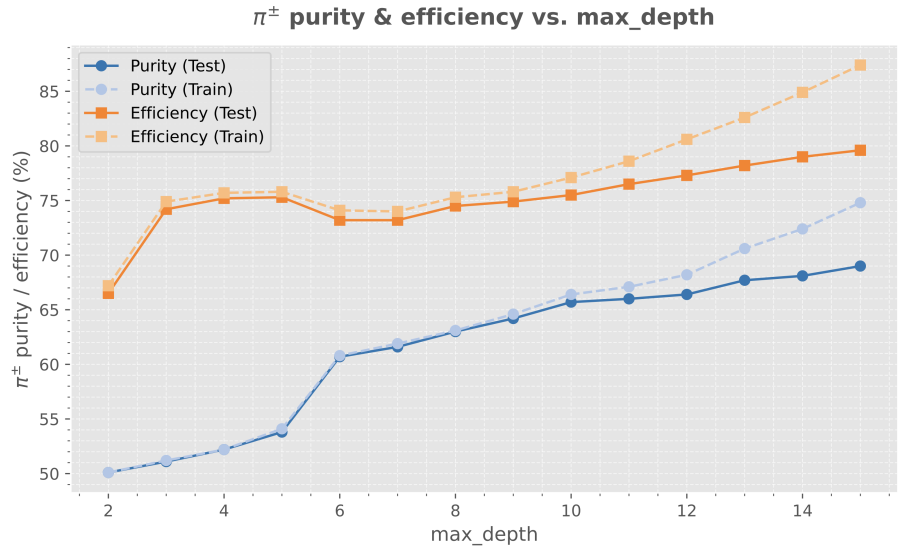

Figure 4: Purity and efficiency vs. `max_depth` for a Decision Tree. The performance on the training set begins to diverge from the training set around a maximum depth of 8, indicating overfitting.

Metric	**Test**	**Train**
Count	73558	294232
Purity (%)	63.3 ± 0.4	64.3 ± 0.2
Efficiency (%)	76.0 ± 0.4	77.7 ± 0.2
Purity $\times$ Efficiency (%)	48.1 ± 0.7	50.0 ± 0.4

Table 2: $\pi^{\pm}$ selection performance for a Full Classification Decision Tree with `max_depth=8`.

Metric	**Test**	**Train**
Count	73558	294232
Purity (%)	70.3 ± 0.4	72.4 ± 0.2
Efficiency (%)	67.3 ± 0.4	69.9 ± 0.2
Purity $\times$ Efficiency (%)	47.3 ± 0.8	50.6 ± 0.4

Table 3: $\pi^{\pm}$ selection performance for a Binary Classification Decision Tree with `max_depth=8`.

Metric	**Test**	**Test (balanced)**	**Train (balanced)**
Count	73558	28316	113262
Purity (%)	52.6 ± 0.3	82.1 ± 0.3	84.0 ± 0.1
Efficiency (%)	89.7 ± 0.3	88.3 ± 0.3	90.3 ± 0.1
Purity $\times$ Efficiency (%)	47.2 ± 0.6	72.5 ± 0.6	75.8 ± 0.3

Table 4: $\pi^{\pm}$ selection performance for a Balanced Binary Classification Decision Tree with `max_depth=8`.

For the full classification model, the confusion matrix (Figure 5) shows that, as well as pions, it performs well at classifying protons, photons, and muons. We also see that the largest source of pion misclassification is from secondary pions ($\pi^{\pm}$:2nd). The binary classification model (Table 3) achieves a higher purity at the cost of efficiency, as its task is simpler. The balanced

binary model (Table 4) shows excellent performance on balanced test data but, as expected, its purity drops significantly when applied to the realistically imbalanced full test set, though its purity remains the highest of the three models.

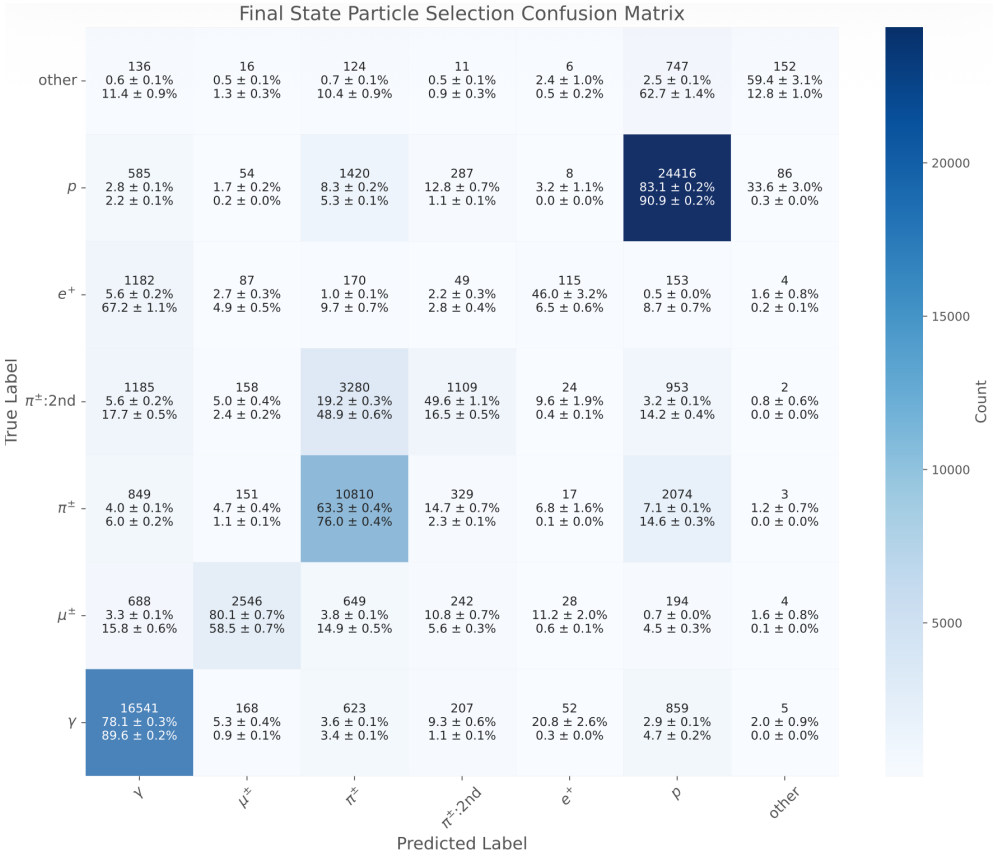

Figure 5: Confusion Matrix for a Full Classification Decision Tree with `max_depth=8`. In each box, it shows the number of predictions, purity and efficiency, respectively.

6.2 Random Forest

Figure 6 reveals something interesting. Aside from early fluctuations, the performance plateaus immediately, regardless of the number of trees. So for our classification problem, a Random Forest of DTs with `max_depth=8` offers no significant performance increase over a single DT. The interpretation of this is that additional trees in the forest are making highly correlated predictions. However in theory, the RF should allow us to increase the maximum depth of the DTs slightly further without overfitting - and that is what we see in Figure 7. Using `max_depth=10` the performance of the RF is shown in Tables 2, 6 and 7.

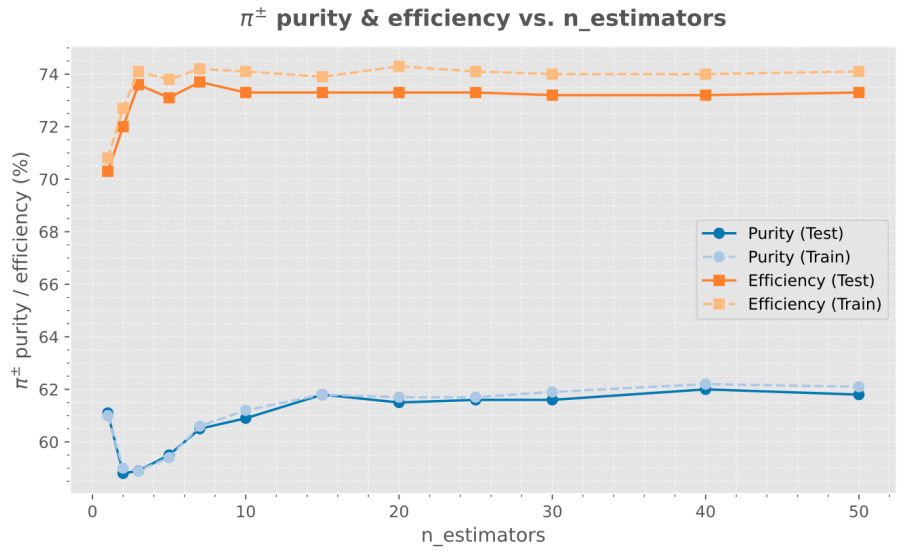

Figure 6: Purity and efficiency vs. `n_estimators` for a Random Forest. Performance does not improve with more trees, indicating a single tree with `max_depth=8` is sufficient.

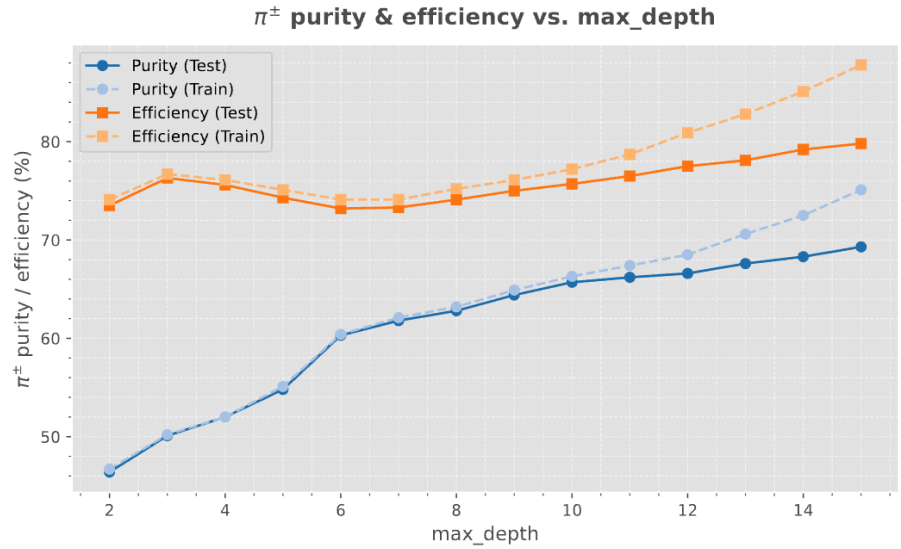

Figure 7: Purity and efficiency vs. `max_depth` for a Random Forest with `n_estimators=50`. We get overfitting at roughly `max_depth=10`

.

Metric	**Test**	**Train**
Count	73558	294232
Purity (%)	65.7 ± 0.4	66.3 ± 0.2
Efficiency (%)	75.7 ± 0.4	77.2 ± 0.2
Purity $\times$ Efficiency (%)	49.8 ± 0.7	51.1 ± 0.4

Table 5: $\pi^{\pm}$ selection performance for a Full Classification Random Forest with `max_depth=10` and n_estimators=50.

Metric	Test	Train
Count	73558	294232
Purity (%)	74.8 ± 0.4	75.7 ± 0.2
Efficiency (%)	60.6 ± 0.4	62.7 ± 0.2
Purity $\times$ Efficiency (%)	45.3 ± 0.8	47.5 ± 0.4

Table 6: $\pi^{\pm}$ selection performance for a Binary Classification Random Forest with `max_depth=10` and n_estimators=50.

Metric	Test	Test (balanced)	Train (balanced)
Count	73558	28316	113262
Purity (%)	55.0 ± 0.3	83.6 ± 0.3	84.9 ± 0.1
Efficiency (%)	91.1 ± 0.2	89.7 ± 0.3	91.6 ± 0.1
Purity $\times$ Efficiency (%)	50.1 ± 0.6	75.0 ± 0.6	77.8 ± 0.3

Table 7: $\pi^{\pm}$ selection performance for a Balanced Binary Classification Random Forest with `max_depth=10` and n_estimators=50.

6.3 XGBoost

Tuning the XGBoost (XGB) model, we see from Figure 8 that overfitting begins after `max_depth=5`. We then optimise the learning rate ("eta") and the number of trees, as shown in Figures 9 and 10.

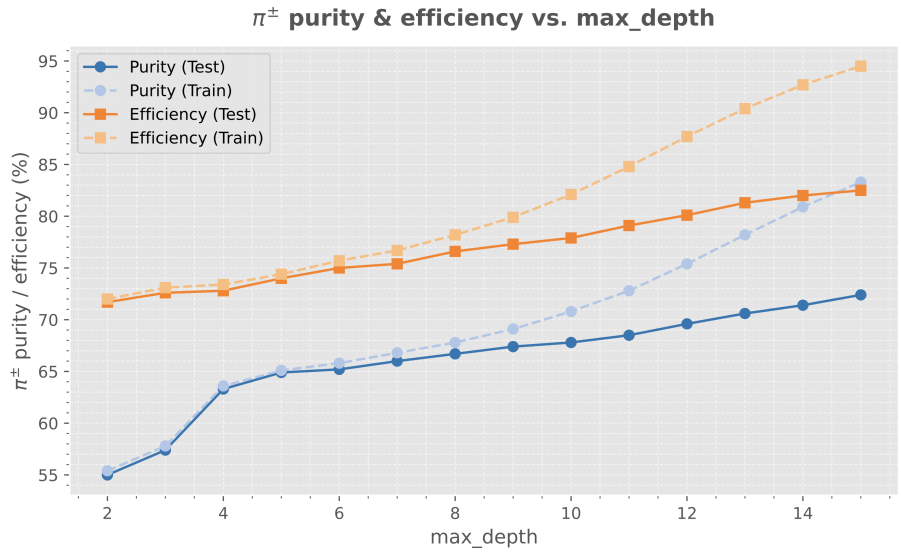

Figure 8: Purity and efficiency vs. `max_depth` for a default XGBoost classifier (100 trees). Overfitting becomes apparent after a depth of 5.

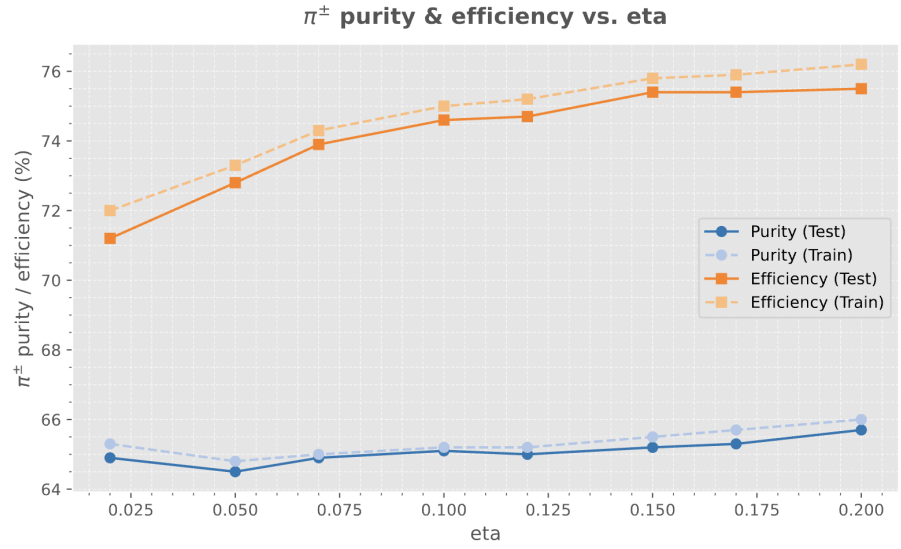

Figure 9: Purity and efficiency vs. learning rate for default XGBoost classifier (100 trees) with `max_depth=5`. Further optimisation is possible without significant overfitting, but our models train quick enough so we stick with `eta=0.2`.

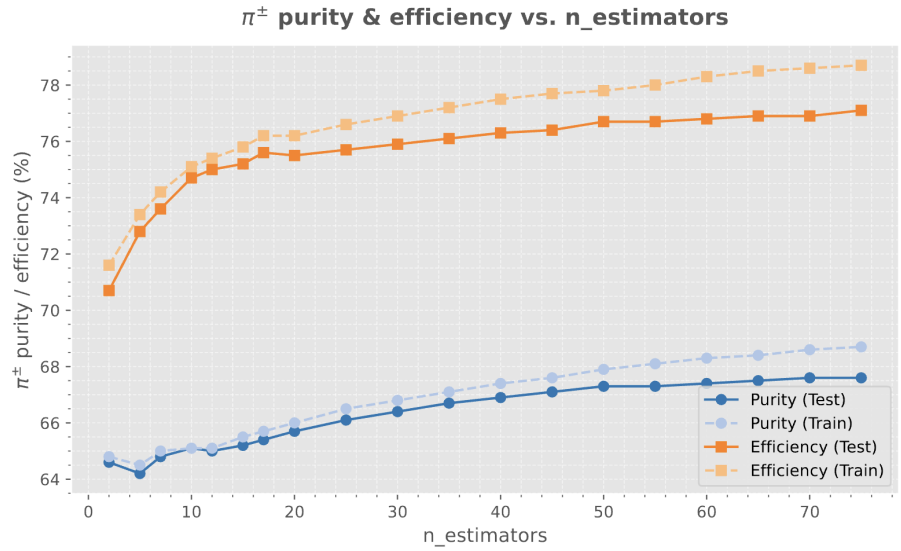

Figure 10: Purity and efficiency vs. number of trees for an XGBoost classifier with `max_depth=5`. Overfitting becomes prominent after about 20 trees.

To further optimise, we implemented a custom loss function designed to aggressively penalise low-confidence or incorrect predictions while rewarding high-confidence, correct pion classifications. Using optimal hyperparameters of `max_depth=5`, `eta=0.2`, `n_estimators=50`, and our custom loss function, we achieve the results in Tables 8, 9, and 10.

Metric	**Test**	**Train**
Count	73558	294232
Purity (%)	67.8 ± 0.4	68.0 ± 0.2
Efficiency (%)	73.9 ± 0.4	74.6 ± 0.2
Purity $\times$ Efficiency (%)	50.1 ± 0.7	50.7 ± 0.4

Table 8: $\pi^{\pm}$ selection performance for an optimised Full Classification XGBoost model.

Metric	Test	Train
Count	73558	294232
Purity (%)	73.4 ± 0.4	74.1 ± 0.2
Efficiency (%)	61.6 ± 0.4	62.6 ± 0.2
Purity $\times$ Efficiency (%)	45.2 ± 0.8	46.4 ± 0.4

Table 9: $\pi^{\pm}$ selection performance for an optimised Binary Classification XGBoost model.

Metric	Test	Test (balanced)	Train (balanced)
Count	73558	28316	113262
Purity (%)	56.2 ± 0.3	84.2 ± 0.3	85.2 ± 0.1
Efficiency (%)	85.7 ± 0.3	85.1 ± 0.3	85.8 ± 0.1
Purity $\times$ Efficiency (%)	48.1 ± 0.6	71.7 ± 0.6	73.1 ± 0.3

Table 10: $\pi^{\pm}$ selection performance for an optimised Balanced Binary Classification XGBoost model.

6.4 ROC Curve

So far, the models predictions for pions are based on whether their probabilities for pions exceed a threshold of 0.5. To visually compare the models' performance across *all* thresholds, we use a Receiver Operating Characteristic (ROC) curve (Figure 11), which plots the True Positive Rate (efficiency) against the False Positive Rate (fraction of background misclassified as signal), defined in Equations 6 and 7 respectively. The Area Under the Curve (AUC) provides a single metric for comparison. The models use the optimised hyperparameters we have found: the DT has `max_depth=8`; the RF has `max_depth=10` and `n_estimators=50`; the XGB has `max_depth=5`, `n_estimators=20` and `eta=0.2`.

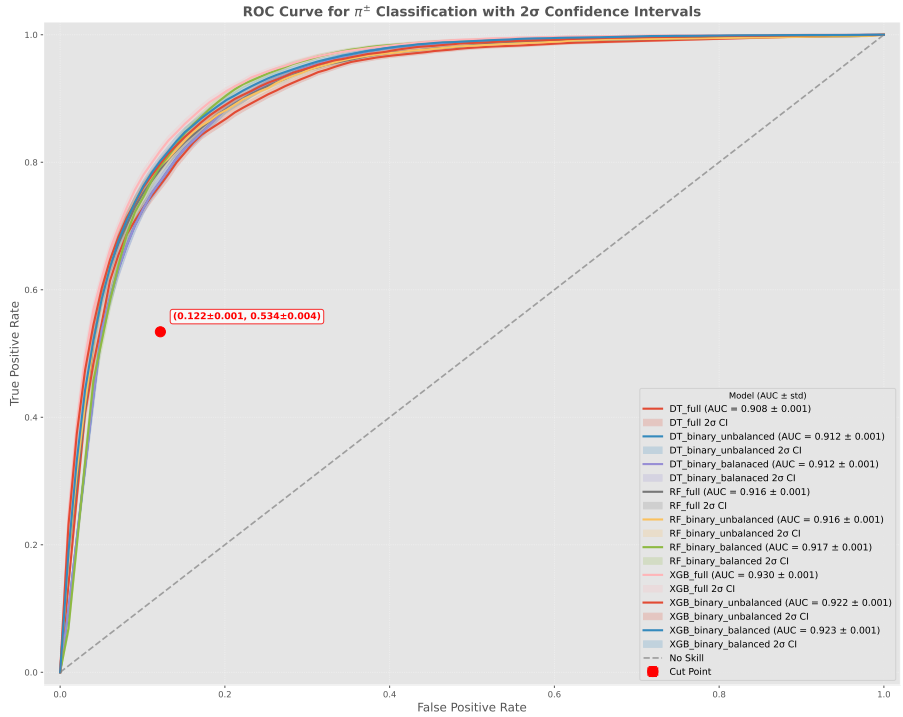

Figure 11: ROC curves comparing the performance of the ML classifiers. The True Positive Rate (Efficiency) is plotted against the False Positive Rate. The diagonal line represents a random classifier (AUC=0.5). The cut-based method represents a single point on this plane.

$$\text{True Positive Rate} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} \tag{6}$$

$$\text{False Positive Rate} = \frac{\text{False Positives}}{\text{False Positives + True Negatives}} \tag{7}$$

7 Discussion

We have successfully demonstrated that taking an ML approach to the classification of final-state $\pi^{\pm}$ in pion-argon interactions yields a significant increase in both the purity and efficiency of the resulting selections.

In Bhuller's thesis, after the $\pi^{\pm}$ selection achieves $51.0 \pm 0.9\%$ purity, a background subtraction step is required before the FSI counts can be made [1]. This is performed using a likelihood fit where MC truth data creates a "template" to estimate and subtract misidentified backgrounds. This subtraction inherits large uncertainties. By improving the initial purity by 15-20% for a threshold of 0.5 (can achieve higher purity for higher thresholds), our ML approach reduces the reliance on this background correction and its inherent uncertainty.

Furthermore, the cut-based method's 53.6% efficiency means that nearly half of the true pion events are discarded. By increasing this by 20+%, our model retains a much larger statistical sample, also reducing the statistical uncertainty in the final cross-section measurement.

An observation was the trade-off between different classification strategies. A binary classifier could be optimised for high purity, since the model is able to focus more on the defining features of pions. In contrast, a multi-class model achieves lower purity at the cost of also being able to distinguish between pions, protons, photons, etc.

Across all thresholds, the best performing model (from a ROC curve perespective) was the full classification XGBoost model. But again, which model to choose depends on the goals of the classification. If high purity is the aim, generally the unbalanced binary XGB model is the best. Whereas for efficiency, the balanced version would be optimal.

8 Conclusion

In this project, we confirmed that tree-based ensembles, namely Decision Trees and Gradient-Boosted Trees, substantially outperform the established cut-based selection for final-state $\pi^{\pm}$ identification in simulated ProtoDUNE data. At a probability threshold of 0.5, our best model (highest purity times efficiency) achieves a purity of 67.8 ± 0.4 and an efficiency of 73.9 ± 0.4, which is a significant improvement over the 51.0% purity and 53.6% efficiency of the cut-based method, thus reducing statistical uncertainties in subsequent cross-section measurements. Based on these successful results, we have several possibilities for future work, including:

- More extensive hyperparameter optimisation.
- Testing of new observables or alternative representations of existing ones.
- Application of these trained models to actual ProtoDUNE experimental data.
- Exploration of different models such as neural networks.
- Applying our ML pipeline to different energy ranges.

A 1D Distributions

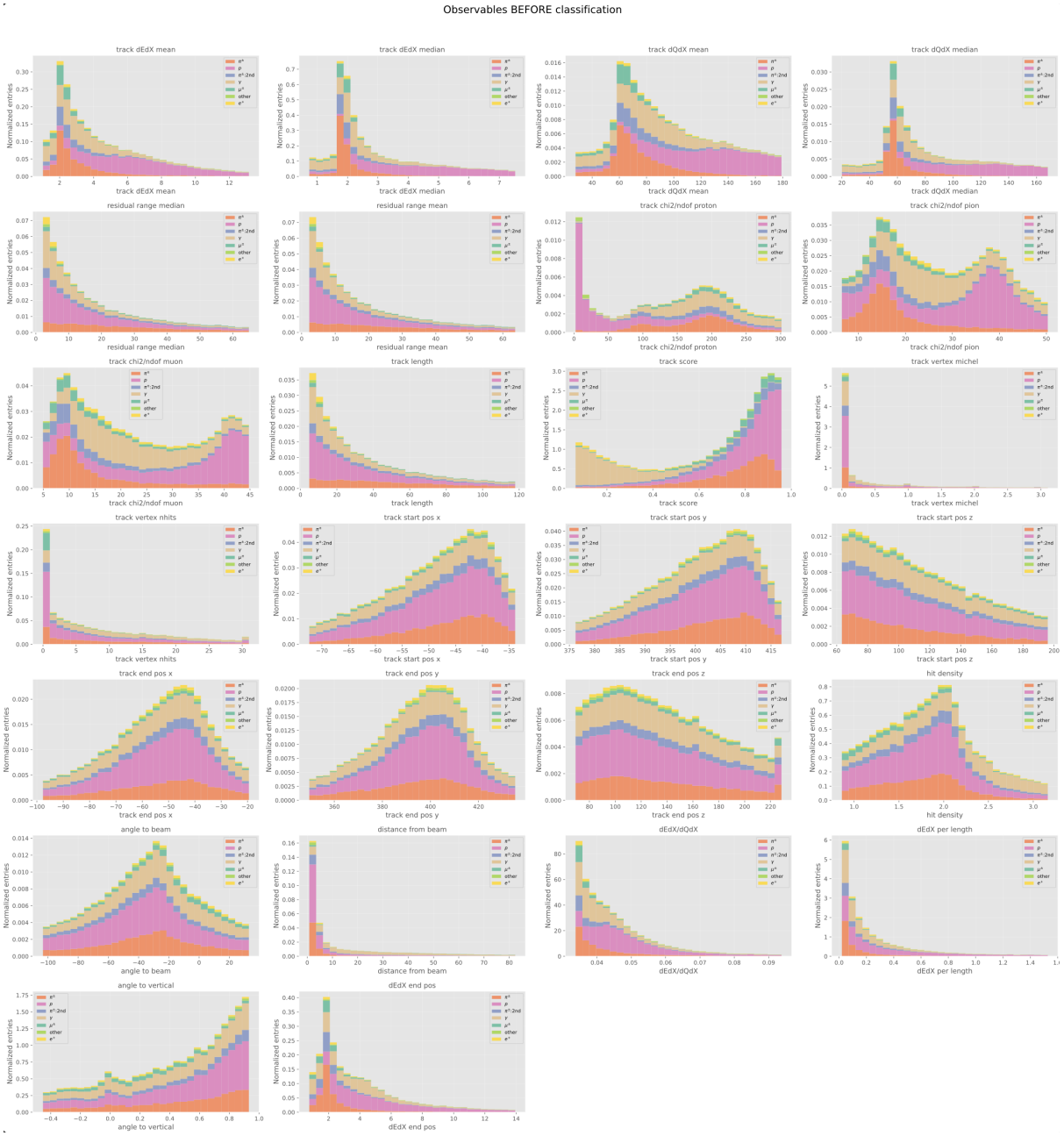

Figure 12: The 1D histograms of the observables before any classification is made.

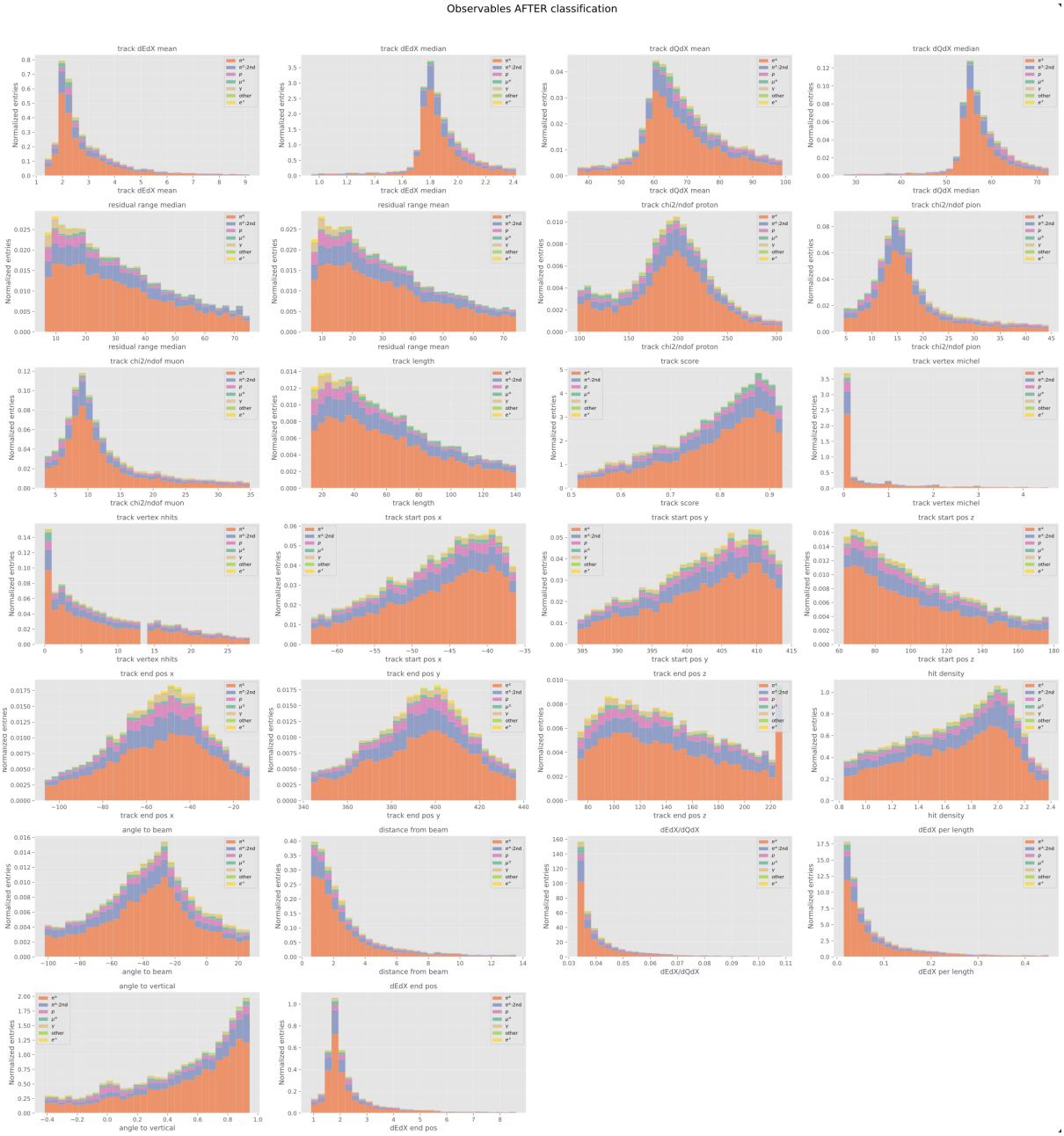

Figure 13: The 1D histograms of the observables a model makes its classifications - in this case a Full Classification Decision Tree with `max_depth=8`).

B Feature Importances

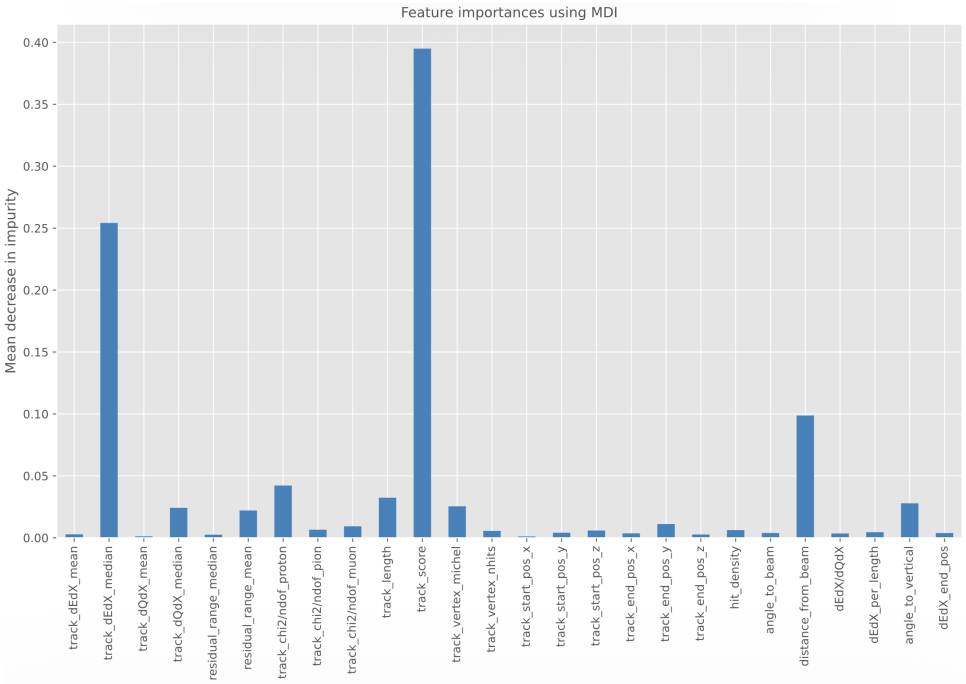

Figure 14: Example plot of feature importances for a Full Classification Decision Tree with `max_depth=8`.

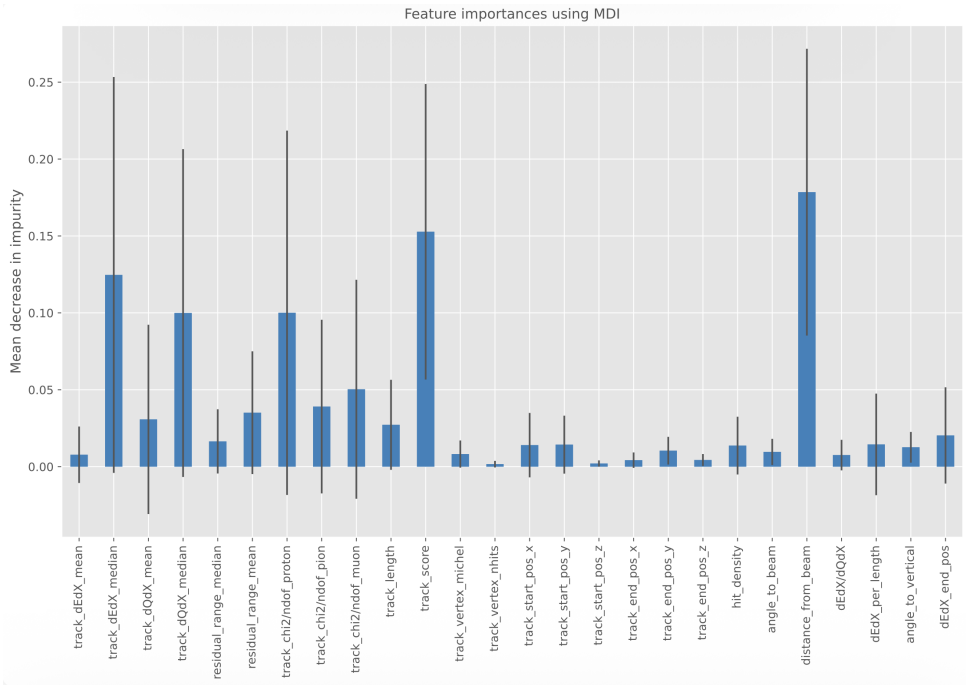

Figure 15: Example plot of feature importances for a Binary Classification Random Forest with `max_depth=10`.

References

[1] S. S. Bhuller, "2GeV π^+-Ar Exclusive Cross Section Measurement at ProtoDUNE," Ph.D. thesis, University of Bristol, 2024.

[2] Scikit-learn developers, "1.10. Decision Trees," scikit-learn 1.5.0 documentation. [Online]. Available: `https://scikit-learn.org/stable/modules/tree.html`

[3] Scikit-learn developers, "sklearn.ensemble.RandomForestClassifier," scikit-learn 1.5.0 documentation. [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html`

[4] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016, pp. 785-794.

[5] XGBoost developers, "XGBoost Models," XGBoost 2.0.3 documentation. [Online]. Available: `https://xgboost.readthedocs.io/en/stable/tutorials/model.html`