Rare Kaon Decay Classifier

Alex Carbo, Sarvan Gill, Jack Deng, Josh Zindler

Project Sponsors:

Dr. Doug Bryman - Project Lead

Dr. Wojtek Fedorko - Machine Learning Expert

Dr. Bob Velghe - System Expert

ENPH 459 - Project 2108

April 14th, 2021

Executive Summary

The theoretical frequency of the extremely rare kaon decay $K^{+} \rightarrow \pi^{+} + \nu \ + \ \bar{\nu}$ (PNN) is yet to be experimentally confirmed. The NA62 detector in Switzerland is attempting to measure this decay. Their goal is to find a way to reliably distinguish between signal and background decay vertices. We employed machine learning techniques including a boosted decision tree and neural network in attempts to improve on previous architectures (TMVA) and accurately classify decay event data. This resulted in our boosted decision tree (BDT) performing better than the raw TMVA results but worse than the quoted TMVA results. Our neural network shows strong promise for differentiating signal and background events, but requires further development before publishing.

Table of Contents

Title page

Executive Summary **1**

Table of contents **2**

Table of figures **3**

Introduction **4**

Theory and Fundamentals **6**

Physics Theory 6

ML Theory 9

Experiments and Results **10**

Boosted Decision Tree Approach 10

Neural Net Approach 16

Conclusions **22**

Recommendations **23**

Deliverables **23**

Appendices **25**

References **27**

Table of Figures

# Title **Pg #**
1 - The Standard Model of Particle Physics …………………………………………………………. 4
2 - PNN Decay Diagram ………………………………………………………………………….. 4
3 - Kaon Decay Vertex …………………………………………………………………………. 5
4 - NA62 Detector CAD ………………………………………………………………………... 6
5 - KTag/GTK/Trim5 ………………………………………………………………………….. 6
6 - Decay Region and STRAWs ……………………………………………………………… 7
7 - STRAW Detector Visualization …………………………………………………………… 7
8 - Decay Vertex Signal Event Diagram ……………………………………………………... 8
9 - Background Event Diagram ……………………………………………………………….. 9
10 - Boosted Decision Tree Visual ………………………………………………………….. 10
11 - Log Scaled ROC Curve for Different n_estimators …………………………………… 13
12 - Log Scaled ROC Curve for Different max_depth …………………………………..… 13
13 - XGB Log Loss Throughout Training …………………………………………………… 14
14 - Log Scaled Early Stop ROC Curve For In and Out of Time Data …………………... 15
15 - Log Scaled Complete Training ROC Curve for In and Out of Time Data ………….… 15
16 - Zoomed In ROC Curve for 3 Different Models ………………………………………... 16
17 - Zoomed In ROC Curve for Current Method …………………………………………... 16
18 - Chamber of Straws with Local Hit Region ……………………………………………. 17
19 - Local Hit Window Visualization (more in Appendix B.1) …………………………….. 18
20 - Neural Net Data Input Diagram ………………………………………………………… 19
21 - PNN Hits/Events Histogram …………………………………………………………….. 20
22 - Model Output vs. Global Features for *pnn_model* …………………………………… 22

Introduction

Project Sponsors:

Dr. Doug Bryman - Project Lead

Dr. Wojtek Fedorko - Machine Learning Expert

Dr. Bob Velghe - System Expert

Our sponsors from TRIUMF tasked us with helping them on their NA62 rare kaon decay project.

Project Motivation

The standard model of particle physics describes the fundamental particles and their interactions. These interactions include when less stable particles decay into more stable particles.

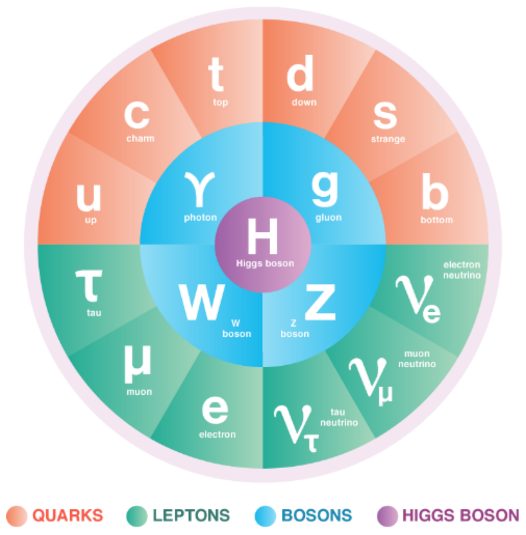


Figure 1: The Standard Model of Particle Physics

In terms of our project, we are concerned with the rare kaon decay $K^+ \rightarrow \pi^+ + \nu + \bar{\nu}$ (PNN) where a kaon decays into a pion and a neutrino/antineutrino pair :

$$K^+ \rightarrow \pi^+ + \nu + \bar{\nu}$$

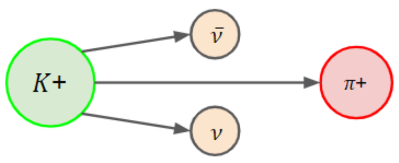


Figure 2: PNN Decay Diagram

This decay (PNN) is extremely rare in that it only occurs once in every *10 billion* kaon decay events. The NA62 detector's goal is to confirm this probability which is predicted by the standard model.

Inside the detector the point in space where the kaon decay occurs is measured by extrapolating the particles' positions and momenta from measurements downstream and upstream to a single point.

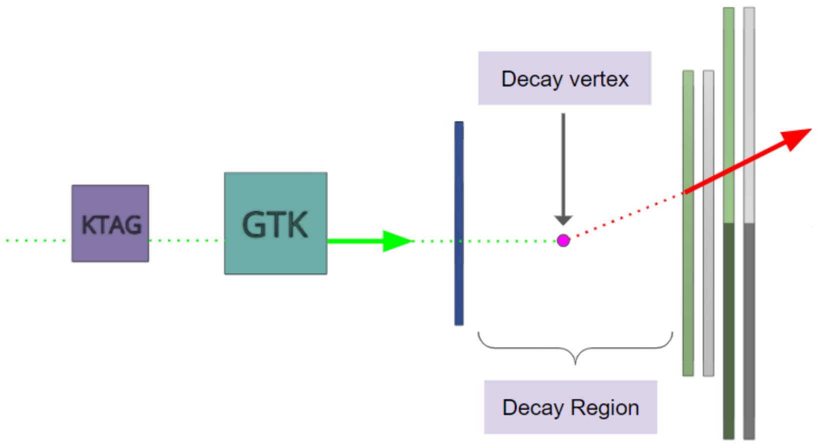


Figure 3: Kaon Decay Vertex

Unfortunately, it is possible that the detector confuses certain events as the decay that we are interested in. This results in there being extra events that are indistinguishable from the PNN decay based on the detector data. We refer to these events as *background* events. By extension, we refer to the event of interest as a *signal* event. We believe that it is possible to design a machine learning model to learn the differences between background events and signal events.

Project Goal

Develop a machine learning model that can distinguish between signal and background for the very-rare PNN kaon decay.

Theory and Fundamentals

Physics Theory

The NA62 Experiment

The NA62 experiment gathers data about incoming particles and determines their types, positions, and momentums. This information is used to determine when a PNN decay has occurred. When repeated enough times, the experiment is able to test the decay frequency.

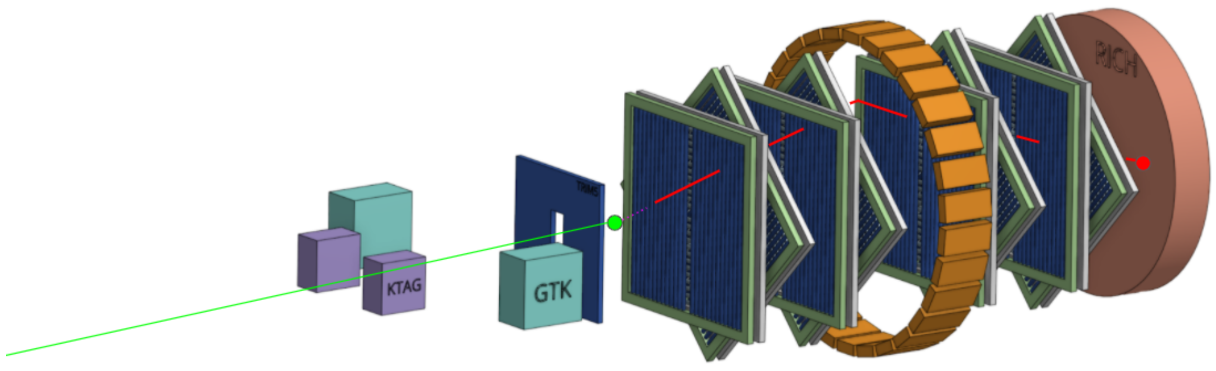
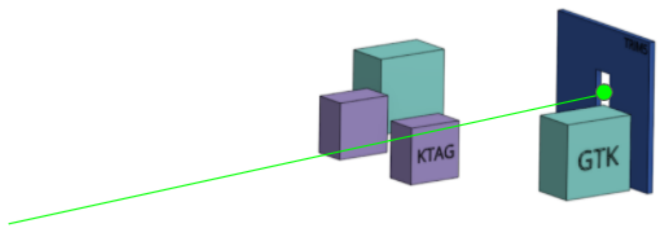


Figure 4: NA62 Detector CAD

To understand the experiment we will walk through a simplified version of the detector step-by-step from the point of view of the particle, explaining what data is measured along the way:

KTag/GTK/Trim5

Initially, a stream of particles of unknown type and momentum travel along the beam path. The KTag detector verifies the particle type as a kaon, and measures it’s time of arrival. The GTK then measures the kaon’s momentum, and the Trim5 Collimator makes the beam path parallel and focused.



Knowns:
- Kaon Time and Momentum

Figure 5: KTag/GTK/Trim5

Decay Region

After passing through the beam collimator, the kaon enters the decay region. This is a large region of empty space where we expect the kaon to decay

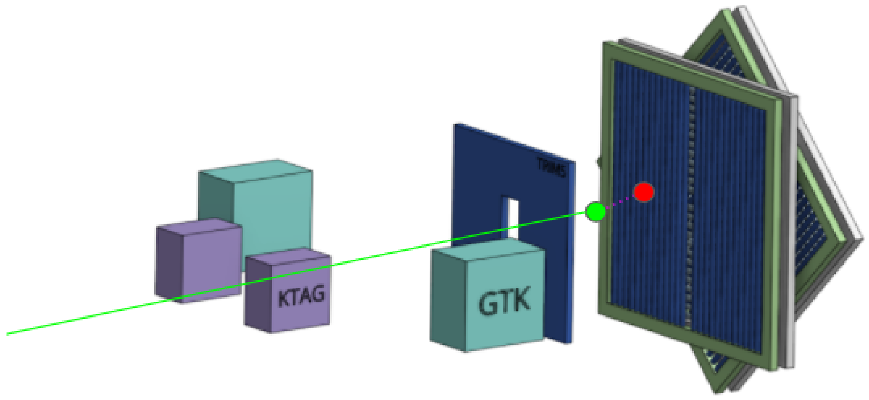


Figure 6: Decay Region and STRAWs

STRAW Detectors

Following the decay region, the now decayed particle passes through what are called STRAW detectors. Here, a mesh of individual “straws” which illuminate when interacted with by particles, are used to measure the particle’s position as shown below. Within the NA62 detector there are 4 chambers of these straw detectors. Each of these chambers contains 4 “views” or orientations where individual straws are placed in parallel. When combining these 4 views a particle position can be reconstructed (Figure 7).

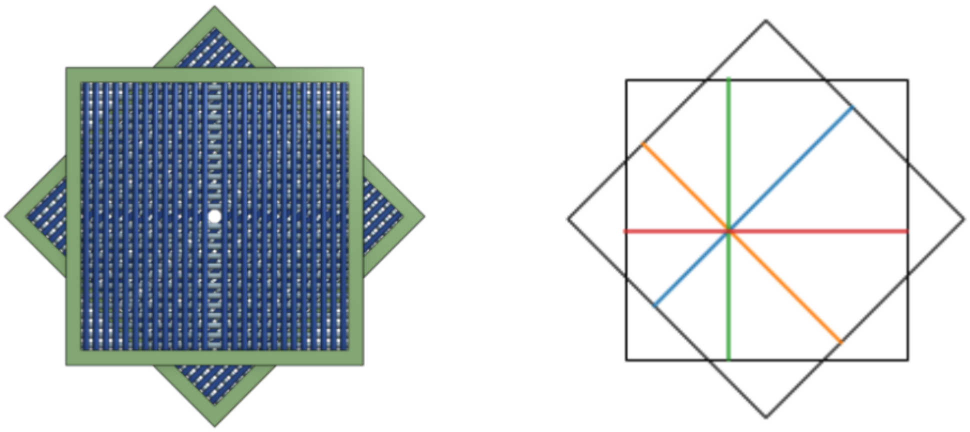


Figure 7: STRAW Detector Visualization

RICH Detector

At the end of the STRAW chambers, the RICH detector receives the decayed particle and verifies its identity. Then, with all this information we can verify that a kaon decayed into a pion, and use the recorded positions and momentums to construct a “decay vertex”.

Decay Vertex

By interpolating the positions and momenta measured in the GTK and STRAW’s, the point in space where the kaon decayed can be found. We call this the decay vertex (Figure 8).

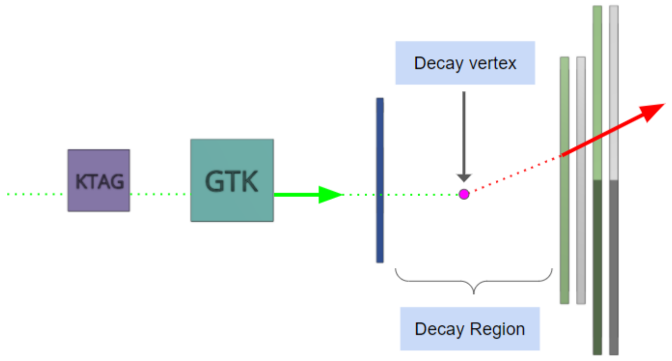


Figure 8: Decay Vertex Signal Event Diagram

Background and Signal

Sometimes, there is ambiguity in the data which can lead to inaccurate conclusions. This can occur if the pion "scatters" when interfacing with the STRAW detectors. When reconstructing the particle's path, the momentum vector created in the straws can be interpolated and matched to the wrong kaon creating a "fake" decay vertex (Figure 9). When this happens, we call this a "*background event*".

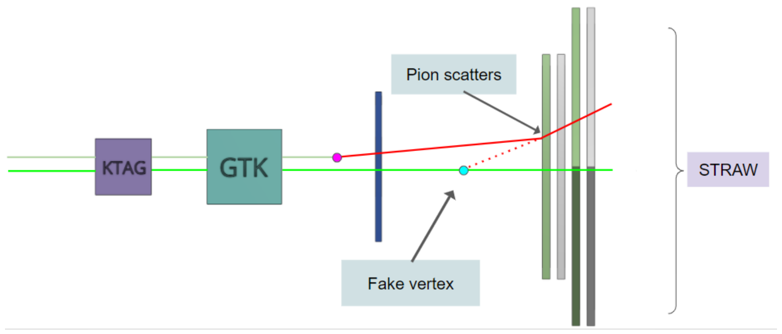


Figure 9: Background Event Diagram

Conversely, any event or data point that *is* the event we are interested in we refer to as the "*signal"* (Figure 8). Since the PNN decay is extremely rare, we do not have access to any substantial PNN signal detector data. This is a very important fact for the project because it means that the data we use as signal data is never real PNN detector data. It forces us to either simulate PNN data, or use other similar decay events as our signal for training.

Machine Learning Theory

Possible Approaches

Boosted Decision Tree (BDT)

A BDT attempts to find "rules" to separate data into various classifications. The decision trees which apply these rules are combined to create a classifier for a set of input data (Figure 10). The trees are positioned in series with one another and focus on improving on the mistakes of the previous tree. When learning, the mistakes of the previous tree are "boosted" so that the next tree will learn from them with higher likelihood.

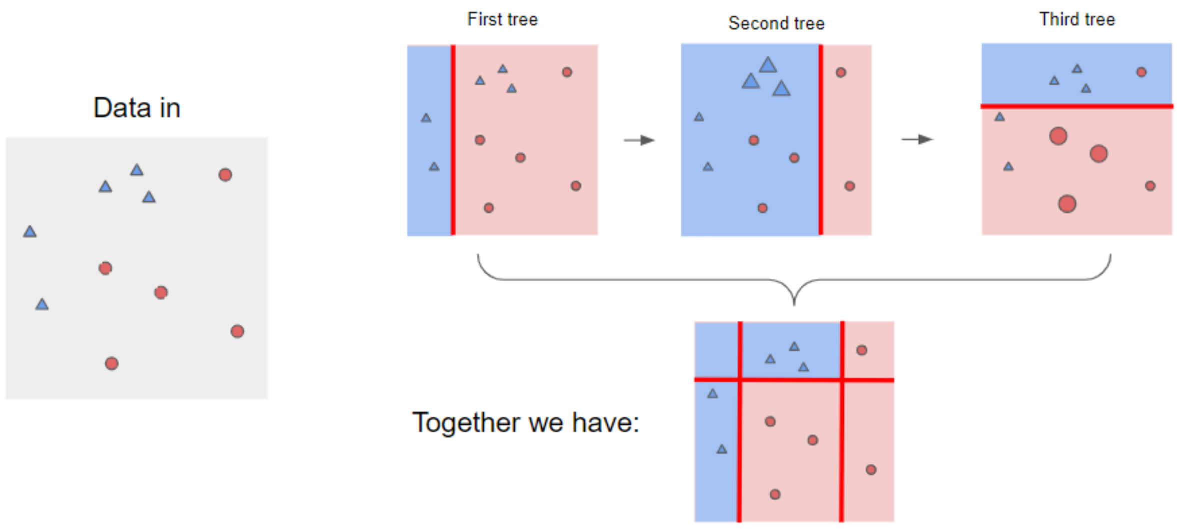


Figure 10: Boosted Decision Tree Visual

Neural Network (NN)
A neural network consists of layers of “neurons” which each have different weights. To learn the data, the neuron weights are adjusted to minimize the differences between the neural net output and training data labels.

Bias/Challenges

Bias is the primary challenge we face in our project. Bias is a property of a machine learning model, referring to when the model learns unintended features. For example, let’s say we want to learn the differences between two objects and one of the objects is simulated in the training data, the model may learn the differences between simulated and not simulated data rather than the actual differences between the objects. This is one type of bias we have had to mitigate in our project.

Experiments and Results

A significant portion of this project was spent investigating bias and issues with the datasets. However, due to access to new datasets and various changes, much of this investigation did not impact our final models. This investigation has been omitted to focus our report on what influenced our results.

Boosted Decision Tree Approach

The existing model that we are attempting to improve upon uses an in-house boosted decision tree (BDT) implementation by CERN. The method is closely related to Adaboosting and its package is called TMVA. In an effort to reproduce their results, we explored similar BDT approaches using currently popular BDT packages like LGBM and XGB. For our project, we focused more on the XGB model and found discrepancies between our classifier output and the current method as presented in their report slides.

Input Data and Datasets

For the BDT, the datasets we used consisted of fields coming directly from our sponsors. These include:

- x,y coordinates of the hit on the trim5 collimator, called “Xtrim5” and “Ytrim5”.
- x,y coordinates of the hit in the first chamber of straws, called “Xstraw1” and “Ystraw1”.
- The hit distance from the center of the first chamber of straws, we call “Rstraw1”.
- x,y,z coordinates of the calculated decay vertex, called “'Xvtx”, “Yvtx” and “Zvtx”.
- The two angles of decay projected in the X plane and Y plane, called “thetaX” and “thetaY”.

The dataset is heavily filtered beforehand. One of the major filters used was time differences. We have 4 sets of data, “out of time” simulated signal data and experimental background data, along with “in time” simulated signal data and experimental background data. The physics behind the difference between in time and out of time data is beyond the scope of this report, but the essential idea is to train our BDT with out of time data and test the model on the in time data.

Previous Method

As explained in the beginning of the section, the current method is using a BDT implemented by TMVA. There isn't clear documentation on how the model is trained or what the hyperparameters used were. Most crucially, we do not have access to their code, thus we cannot reproduce their results directly. However, our sponsors do have access to TMVA, thus they have trained TMVA's BDT with default hyperparameters with our datasets.

We have their report slides with their classifier output histogram and ROC curves to compare ours against which will be discussed more in a later section.

Hyperparameter tuning for BDT

The main parameters we tuned were:

- The number of boosted trees or the number of boosting rounds used, we call "n_estimators".
- The maximum depth we allow each boosted tree to grow to, we call "max_depth".
- The boosting learning rate, we call "learning_rate".

For n_estimators, we tried out from 5, 10, 15, 30, 60, 90, 100 and 150. The one that produced the best ROC curve on the in time data was n_estimators = 60. The more n_estimators there are, the longer the runtime, and yet there is not much improvement in the performance. (ROC graph for n_estimators = 100 and 150 are in the Appendix A.1)

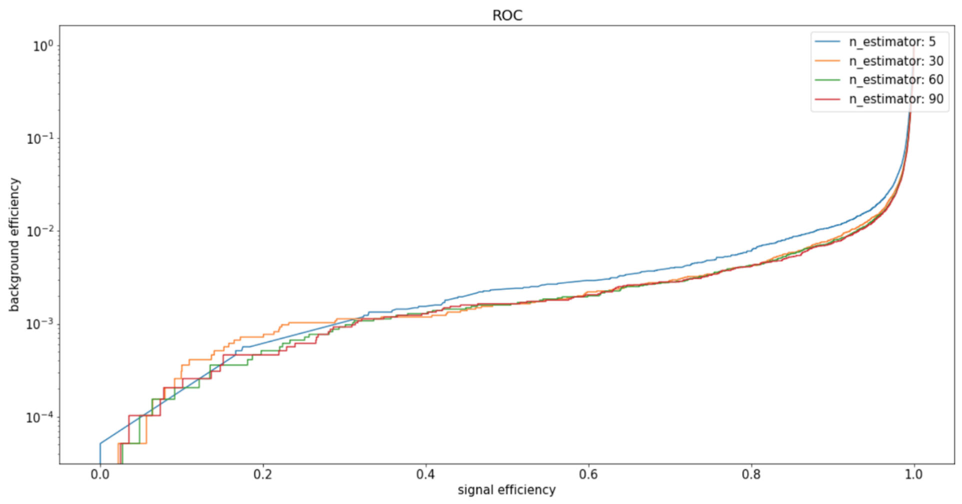


Figure 11: Log Scaled ROC Curve for Different n_estimators

For max_depth, we tried out from 10, 20, 30, 40, 50, 60, 80 and 100. The one that produced the best ROC curve on the in time data was max_depth = 40. Interestingly, it performed very well when signal efficiency was between 0.2 and 0.6. It seems to be the depth that can generalize best onto the in time data. (ROC graph for max_depth = 30 and 50 are in the Appendix A.2)

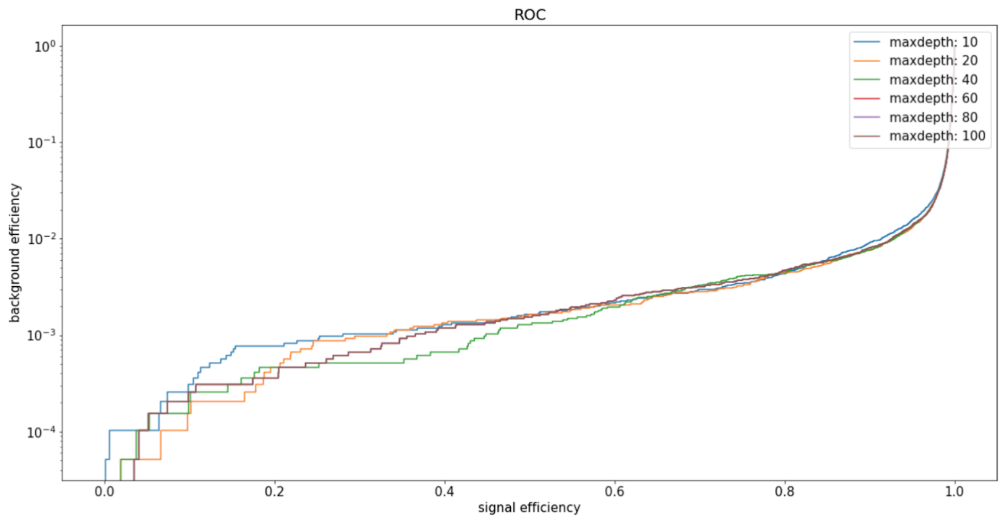


Figure 12: Log Scaled ROC Curve for Different max_depth

For the learning_rate, we tried out from 0.1, 0.3, 0.5. The results were similar between the three with no real improvements on either. We would recommend sticking with the default 0.3.

Overfitting concerns

To prevent overfitting our model, we will be using early stopping. We evaluate the log loss for both the training data (out of time data) and testing data (in time data) on each iteration and stop the training when the log loss on the testing data increases. Here is a plot of our loss throughout a complete training process of 150 iterations:

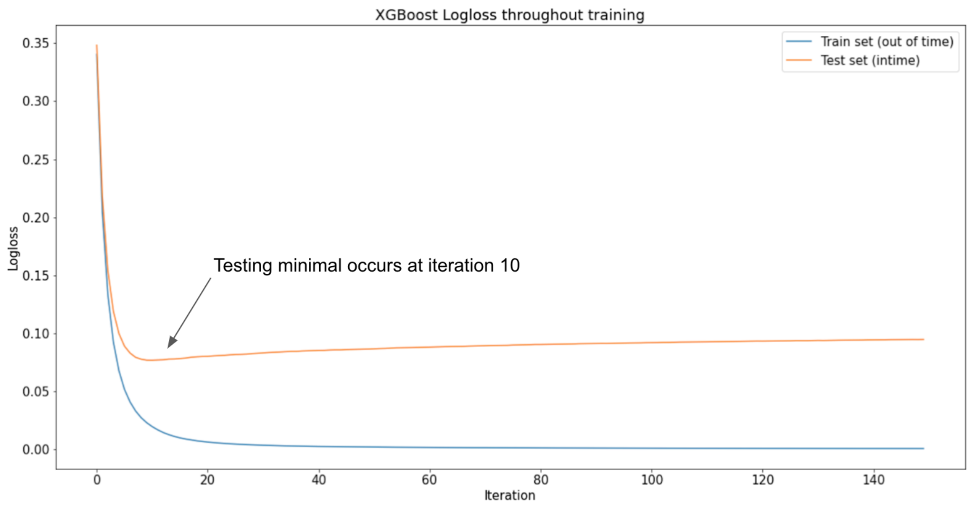


Figure 13: XGB Log Loss Throughout Training

By early stopping at iteration 10 we can be more confident that our model is not overfitting. More work is required to achieve certainty.

Results

Using the early stop model, we achieve an overall accuracy of 98% on the in time data, extremely similar to that of a completely trained model. Interestingly, despite having a lower log loss on the testing set for the early stop model, its binary accuracy is slightly worse (factor of 0.01%) than that of a completely trained model. Here are the ROC curves both for early stop and completely trained.

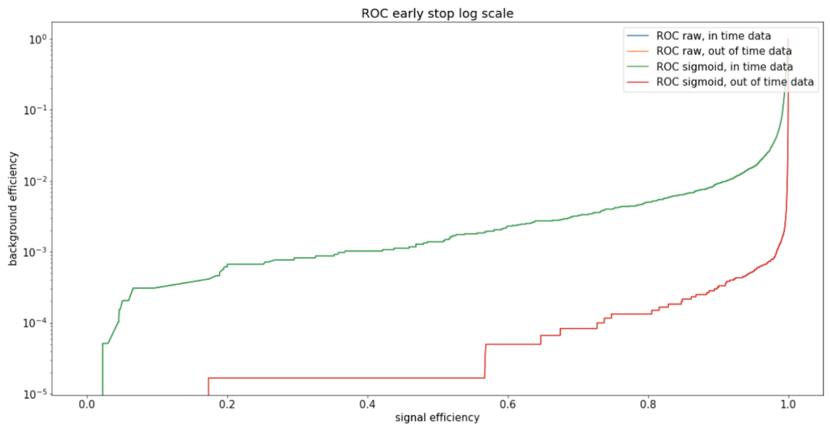


Figure 14: Log Scaled Early Stop ROC Curve for In and Out of Time Data

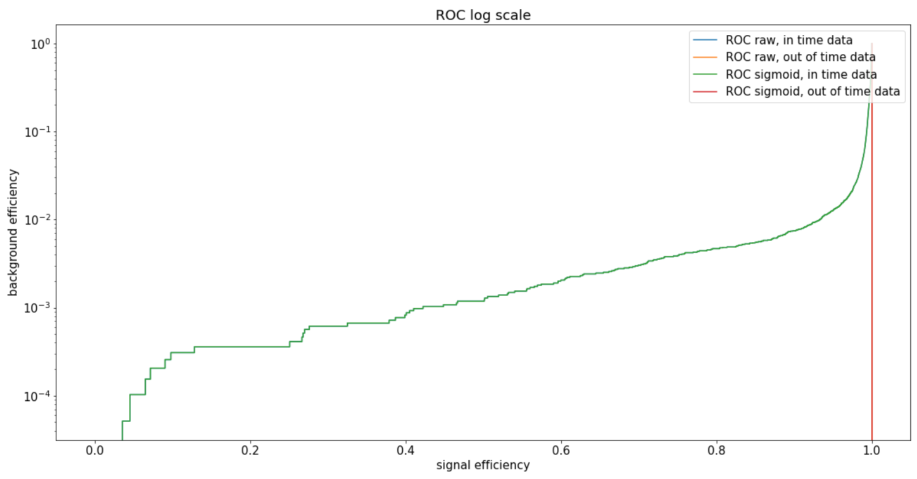


Figure 15: Log Scaled Complete Training ROC Curve for In and Out of Time Data

The classification reports and confusion matrix for the model are included in the Appendix A.3.

Comparison to TMVA output

We can compare our classifier output on the same dataset against the BDT implemented by TMVA, here are the ROC curves zoomed in at background efficiency between 0 and 0.002: (This specific section of the graph was chosen because it was presented in the report slides for the current method and is what we want to compare against. The log scale comparison is included in the Appendix. A.4)

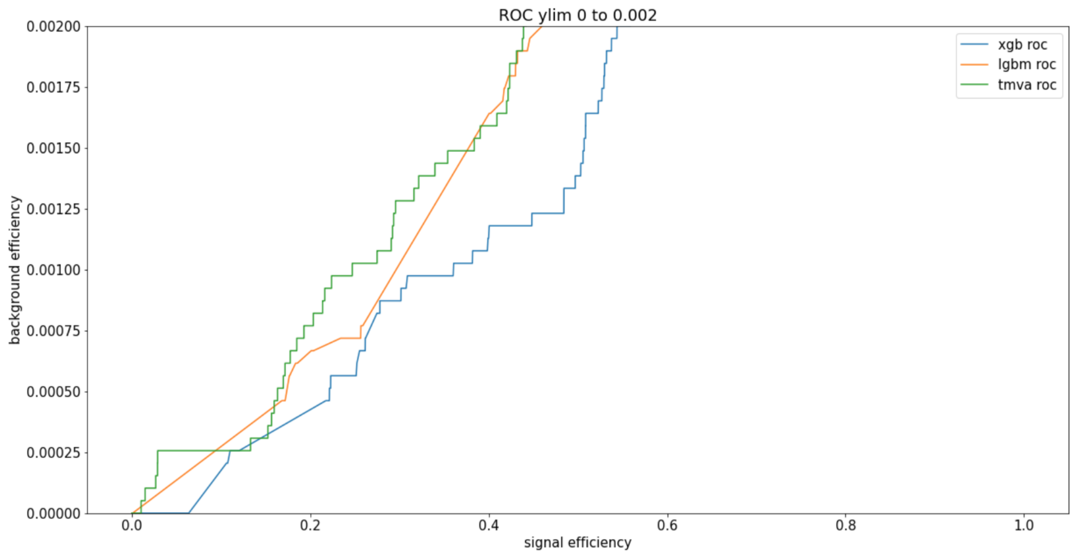


Figure 16: Zoomed In ROC Curve For 3 Different Models

As seen, all three have similar performance at background efficiency = 0.001 with a signal efficiency around 0.3 to 0.4. However if we take a look at the ROC presented in the report slides for the current method:

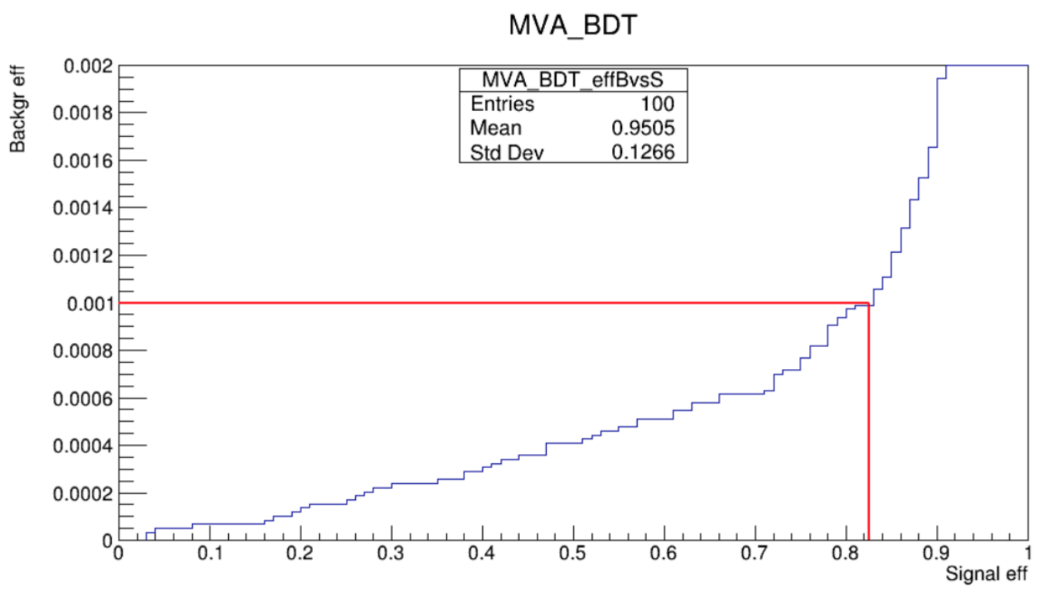


Figure 17: Zoomed In ROC Curve For Current Method

We see that the slides indicate that the current method is performing 2 times better than all three classifiers we tested on our dataset. This draws a conclusion to the BDT section. The discrepancy between the current method and our methods needs to be further understood before we can advance for BDT and have confidence in our results. (The histogram for our classifier's and the current method's raw output in Appendix A.5-I and Appendix A.5-II)

Neural Net Approach

An important piece of data that we have access to is all the straws that were hit during a single event. If scattering is indeed one of the reasons for the misclassification of background events as signal events, we would hope to see some indication of this from the straws hit close to the pions path. This data has not been leveraged in any analysis yet so learning anything from these features would be extremely helpful. This motivates the use of a neural net to potentially learn these features.

Local Hit Data

To look for potential signs of scattering or other events we look at the straws in a small region around the pions path. In Figure 18, the colored lines are straws that were hit by particles, the black dot represents the position of the pion in the straw chamber and the red outline represents the small region of interest. We focus only on the straws within this region. The size of this red region is arbitrary and becomes one of the hyperparameters of our neural network.

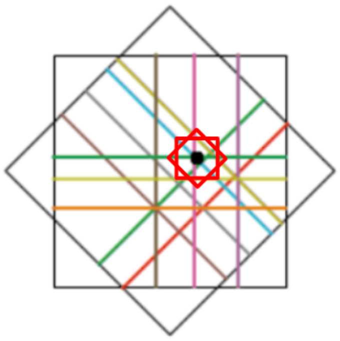


Figure 18: Chamber of straws with local hit region

Because our region is much smaller now, it is useful to look at the views (U, V, X, Y) individually and trace the pion's path through the entire chamber. Looking at the cross section of view and incidentally the cross section of the straws gives a better idea of which straws were hit around the pions path. Figure 19 below illustrates this in a nice diagram. Each subplot represents a single view of straws within a chamber. Recall there are 4 views (U, V, X, Y) in a straw chamber. For each view, the circles in the figure represent singular straws. A black circle represents a straw that was hit by a particle. The red line is the pion's reconstructed path through the detector.

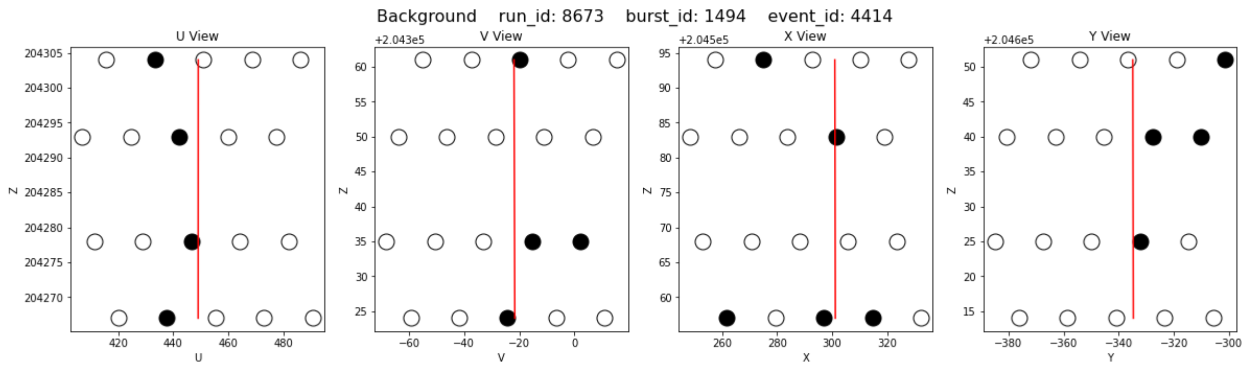


Figure 19: Local Hit Window Visualization (more in Appendix B.1)

Since each view has 4 layers of parallel straws, the height of the diagram is fixed. In other words the straws only lay in fixed z positions. Thus, the size region is solely determined by the width or how many straws we decide to include per layer. We call this number n, and this width will become a hyper parameter of the neural net. In Figure 19, we can see that each row has 5 straws thus $n = 5$ for this example.

These windows are created by selecting the $4n$ closest straws to the pions path in each view. As there are 4 views in a chamber, and 4 chambers in our detector, this gives us a total input size of $64n$.

Framing the Data

Now that we have an idea of what we want to train on, we need to frame the data in such a way that we can use it as an input into a neural net. In order to do this, for every straw in our region of interest we assign it a unique index $i,\ i \in [0,\ 64n)$. Then we create a vector $\vec{v}$ to represent if each straw was hit or not. That is,

$$v_i = \begin{cases} 0 & \text{if straw } i \text{ was not hit} \\ 1 & \text{if straw } i \text{ was hit} \end{cases}$$

By doing this we flatten the images shown in Figure 19, to a single vector which we can then input in our neural net. This process is nicely displayed in Figure 20. The figure shows how we start with a region of interest for each chamber, flatten it into our vector and then finally give that vector to a neural net to make a prediction on whether it is signal or background.

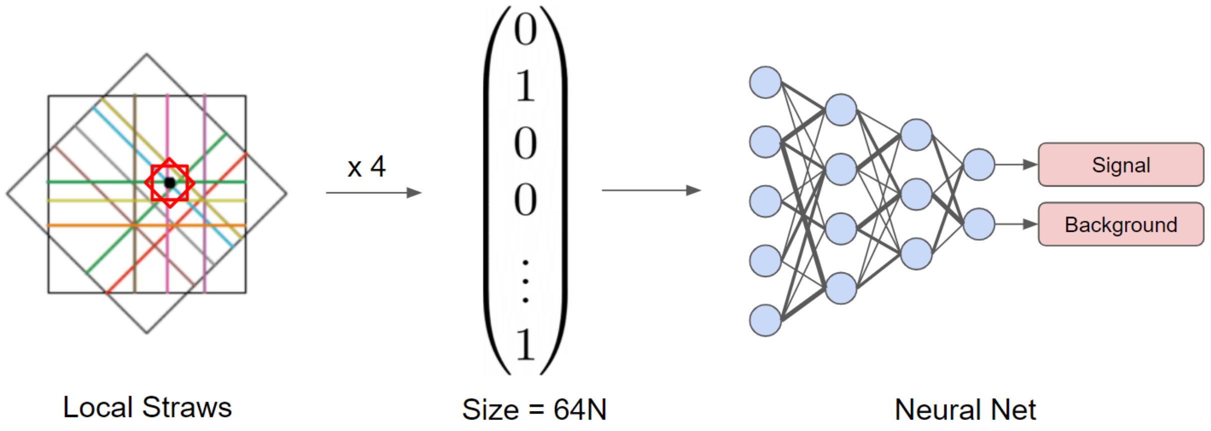


Figure 20: Neural Net Data Input Diagram

Potential Bias

Before we talk about neural nets and its results, it is important to acknowledge where bias could arise in this data.

When we look at our PNN data, we need to be confident that our neural net will not learn artifacts of the simulation. Recall that our signal data for PNN is simulated while our background data is taken from the actual detector. For the local hit window, the most obvious way that the simulation could impact the results is for there to be a different amount of straws hit in the simulation compared to the actual detector. A preliminary analysis for the total number of straw hits is shown below in Figure 21.

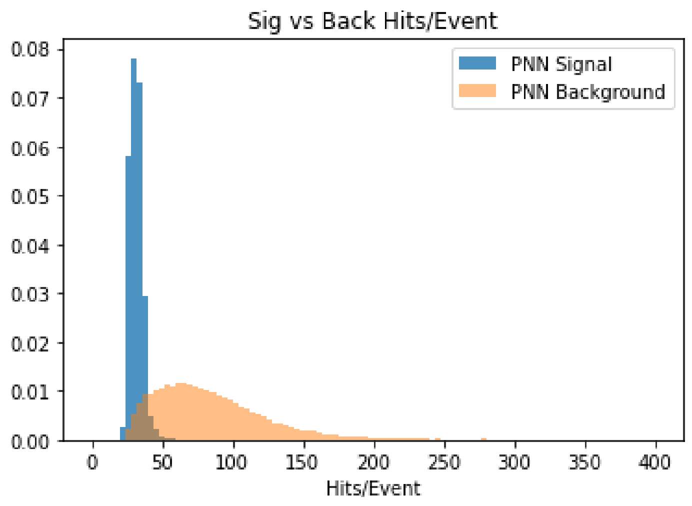


Figure 21: PNN Hits/Event Histogram

This histogram compares the total number of hits for an event between the PNN signal and PNN background. Here we can clearly see that PNN signal has less hits on average when compared to the background. This is a potential bias that our architecture could learn so some filtering should be done on the data to train on events with similar number of hits.

Interestingly, because we are looking just at the straws around the pion's reconstructed path, we are hoping to only learn information about potential scattering due to the straws. This gives us reason to train on another event $K^+ \rightarrow \pi^+ + \pi^0$, referred to as $K_{\pi 2}$. This event is much more frequent so there is an abundance of data for both signal and background events. Although the kinematics of the event are slightly different we expect the π^+ to interact with the straw detectors similarly to the event of interest ($K^{+} \rightarrow \pi^{+} + \nu + \bar{\nu}$). The purpose of using this data is to mitigate the aforementioned bias of learning the simulation, since we have detector data for both $K_{\pi 2}$ signal and $K_{\pi 2}$ background.

However, by mitigating our initial bias between the simulation and the detector, we introduce a new bias that the $K_{\pi 2}$ data does not accurately represent the PNN event in the local hit window. Although, this seems unlikely, it is still something that we must confirm after we have results from our neural net.

Training and Results

The first step after designing the local hit data input was to ascertain whether or not it had any predictive value in classifying signal and background events. This was done using signal and background $K^{+} \rightarrow \pi^{+} + \pi^{0}(K_{\pi 2})$ events. The π^{0}in this decay quickly decays into two photons, and the Liquid Krypton calorimeter (LKr) detects these. The signal selection here requires both of the photons from the kaon decay to be detected in the LKr. The background selection is done exactly the same as it is for PNN events. A very simple neural network (Appendix B.2) was trained with these datasets using the local hit data with a window width $n = 5$(this window width is constant in all described neural networks), achieving around 75% accuracy. However, it is believed that there is an issue with the selection of the background in these events. It is very likely that the selected background is actually a mixture of signal events and background events. The only reason this selection is valid to use for PNN events is because the relative frequency of background to signal events is so high. This is not the case for $K_{\pi 2}$ events, in fact this decay signal is much more frequent than the background. Due to this selection bias we cannot use this data, and therefore our result is meaningless.

Simulated PNN signal and Real PNN background was then received that had been extensively filtered and cut such that there is a very high confidence that there is no bias between them. That is, any difference between the two datasets is characteristic of a difference between signal and background PNN events, not a difference between simulated and real events. We gain this confidence as this method is published, presented, and peer reviewed. It is therefore our assumption that this method is valid and we are left with unbiased datasets. A slightly more interesting NN (Appendix B.3) was used for training on this data. This NN achieved 90% accuracy on our test set, suggesting the local hit data method has quite strong predictive power in classifying signal and background events. This model is called *pnn_model.*

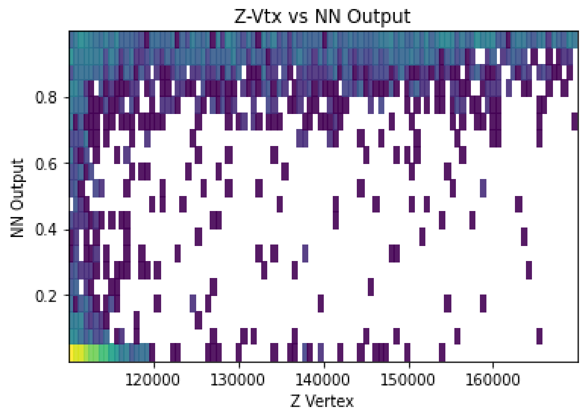
The same model architecture was then trained on our $K_{\pi2}$ signal data and PNN background data, as both of these datasets are from the NA62 detector and not simulation. This model achieves an accuracy of 85% on its test set, which supports our assumption that we are not learning the difference between simulated and real data in our PNN training. Of course, this method introduces a new bias - that is that our model may be learning the difference between $K_{\pi2}$ events and PNN events. This model is called *kpi2sig_pnnbckg_model*.

Influence of Global Features

The hope is that the information gained using the local hit window is independent from the global features used in the BDT approach (momentum, vertex position, etc.), which differ between the $K_{\pi2}$ and PNN events. If this is true, it would allow us to train our model using $K_{\pi2}$ data, and use it on PNN data. As stated previously, $K_{\pi2}$ data is orders of magnitude more numerous than PNN data, which would vastly improve the accuracy of our model.

In order to investigate this, we can plot the output of our neural network versus the global feature for each event in our test set, and try to discern some overall trends, eg: Are a lot of false positives observed in a momentum region more likely to be associated with a signal event? If so, this would suggest that the local hit data is influenced by the global feature.

The plot for model output versus z coordinate of the decay vertex and z component of the pion momentum vector for *pnn_model* are shown below in Figure 22.



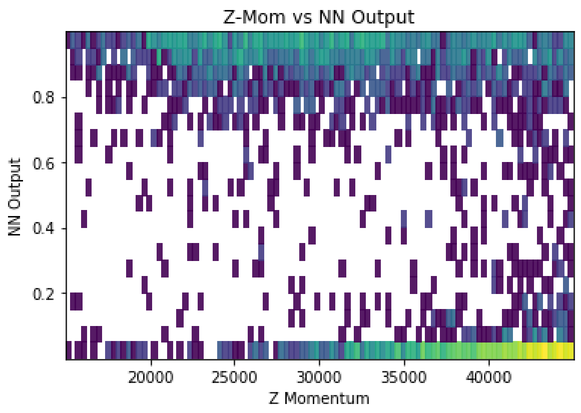


Figure 22: Model Output vs. Global Features for *pnn_model*

No obvious trends emerge from these plots, but further analysis into misclassified events should be explored. The tendency of events to be incorrectly classified when the global features suggest the same incorrect classification can be used to find a sort of covariance between the local hit data method and the influence of the global features. In the absence of correlation, we can confidently use $K_{\pi 2}$ data to train our model, allowing much more input data, yielding a higher accuracy model.

Another benefit of finding this correlation is to determine the usefulness of ensembling our NN and BDT in order to make a stronger model. A lower correlation implies a higher reward from ensembling.

Conclusions

Our XGB BDT performed quite well achieving a 98% accuracy on the in time data and out performing the default LGBM and TMVA output. However, it is noticeably worse when we compare our ROC curves to the current method's as reported on the slides. This is a major roadblock that needs to be investigated, we need to know exactly what filters on the data and hyperparameters were used by the current method. To continue work on the XGB model, we would recommend starting with investigating overfitting. Maybe trying L1, L2 regularization would help. More tuning on the model could also be done once confident there is little to no overfitting.

On the neural network side of the project, promising results have been observed suggesting the local hit data method has strong predictive power in distinguishing signal and background events. 90% accuracy was achieved when training with Simulated PNN signal and Real PNN Background, while 85% accuracy was achieved when training with Real $K_{\pi 2}$ signal and Real PNN background. The first next step is to use *kpi2sig_pnnbckg_model* to classify PNN data, to see if a model trained using $K_{\pi 2}$ data can generalize to PNN. Then, the influence of global features on NN output should be further investigated by observing misclassified events, and determining whether misclassified events would also be misclassified using the global features (a sort of correlation). If this correlation is weak, the local hit data is likely independent of the global features, meaning $K_{\pi 2}$ events can be used for training, and there is a strong case for ensembling our BDT and NN.

Significant progress has been made towards a better decay vertex classification system for the NA62 experiment at CERN. However, as mentioned in the above conclusions, more work needs to be done. To this end, we have created a GitLab repository containing utility functions, jupyter notebooks, and accompanying documentation that make it easy to recreate and expand on our work.

Recommendations

- Investigate TMVA Discrepancies
- Observe whether or not *kpi2sig_pnnbckg_model* generalizes to PNN data
- Find correlation of events misclassified by our neural network and global features to determine usability of $K_{\pi 2}$ data, and usefulness of an ensemble of the BDT and NN

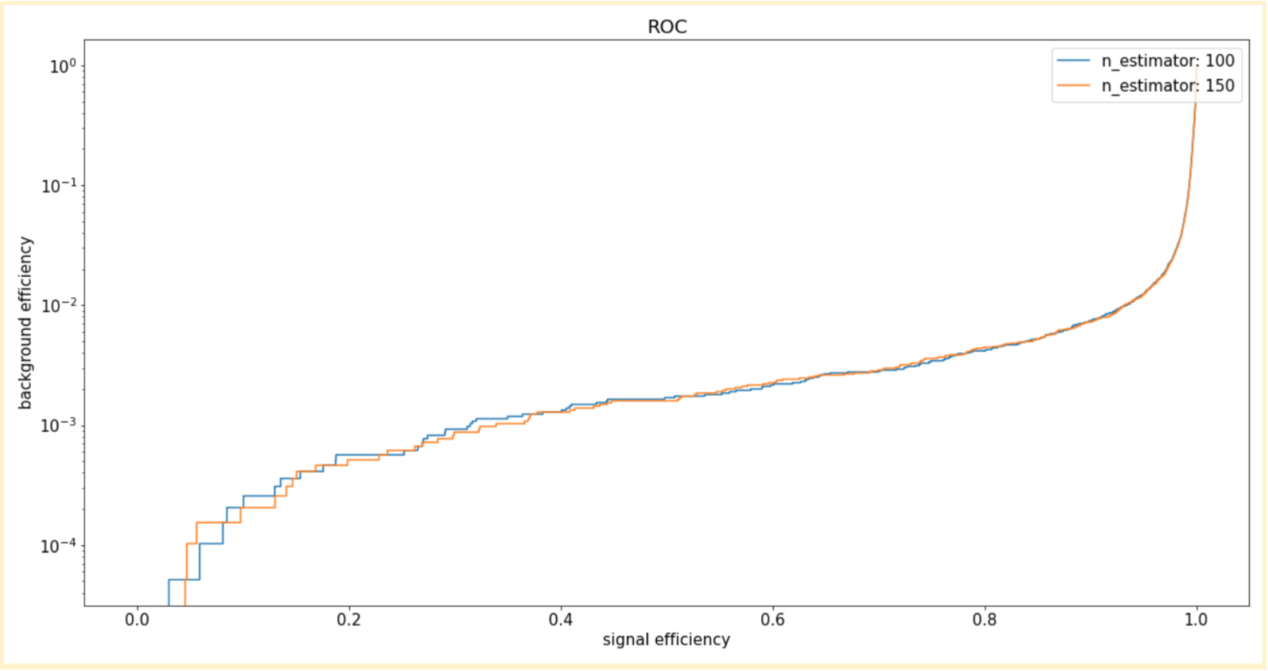
Deliverables

1. Documentation
2. Packaged and cleaned code/models
3. This Report

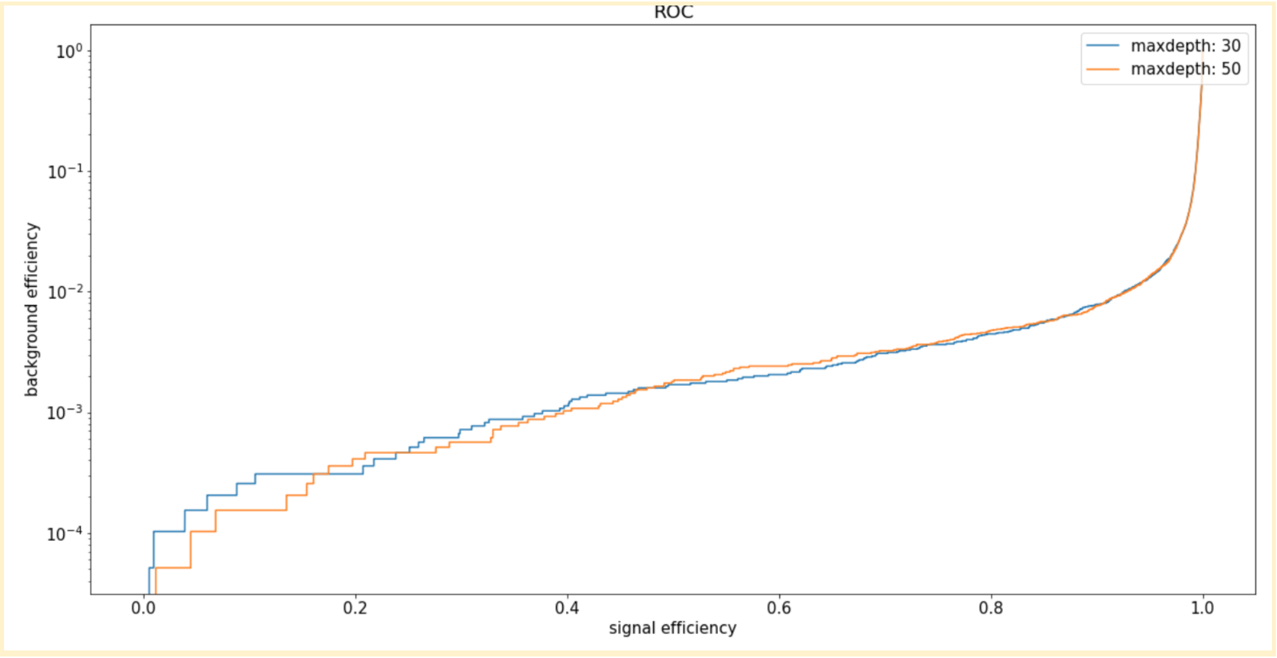
Appendices

Appendix A - Boosted Decision Tree Appendix

Appendix A.1: Log scale ROC curve for when n_estimator = 100 and 150



Appendix A.2: Log scale ROC curve for when max_depth = 30 and 50



Appendix A.3: Classification report and confusion matrix for our XGB model

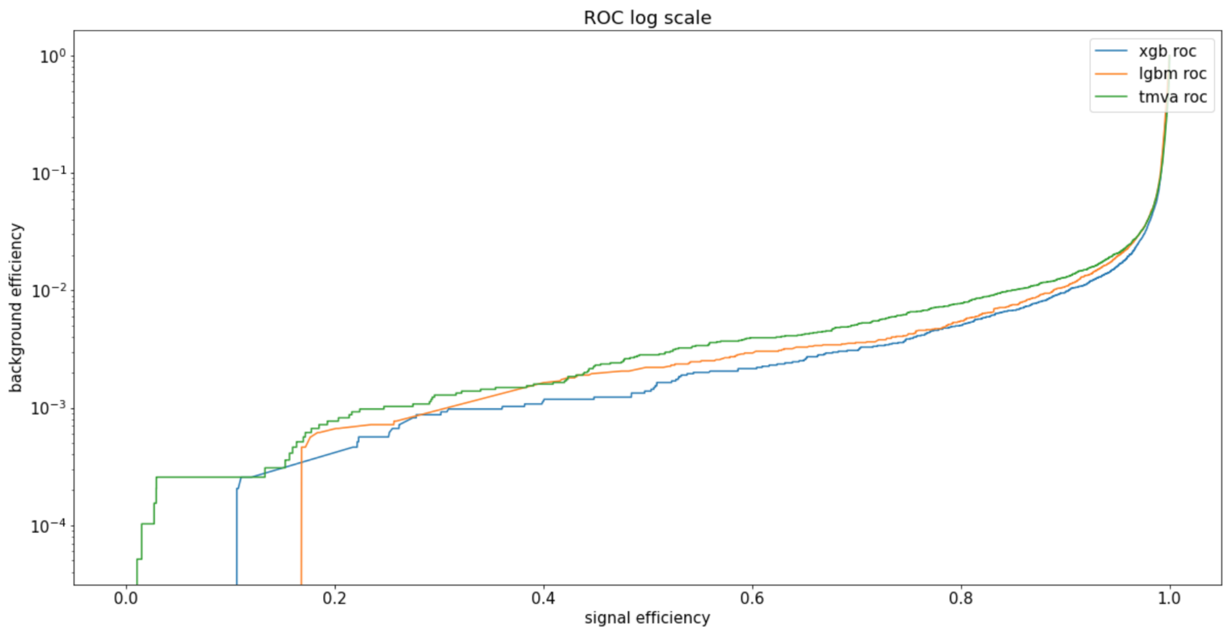
Early stop

```
=== testing results on test data ===
              precision    recall  f1-score   support

         0.0       0.56      0.97      0.71     19483
         1.0       1.00      0.98      0.99    651439

    accuracy                           0.98    670922
   macro avg       0.78      0.97      0.85    670922
weighted avg       0.99      0.98      0.98    670922

[[ 18858    625]
 [ 14593 636846]]

=== testing results on train data ===
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     60028
         1.0       1.00      1.00      1.00     60028

    accuracy                           1.00    120056
   macro avg       1.00      1.00      1.00    120056
weighted avg       1.00      1.00      1.00    120056

[[59844   184]
 [  210 59818]]
```

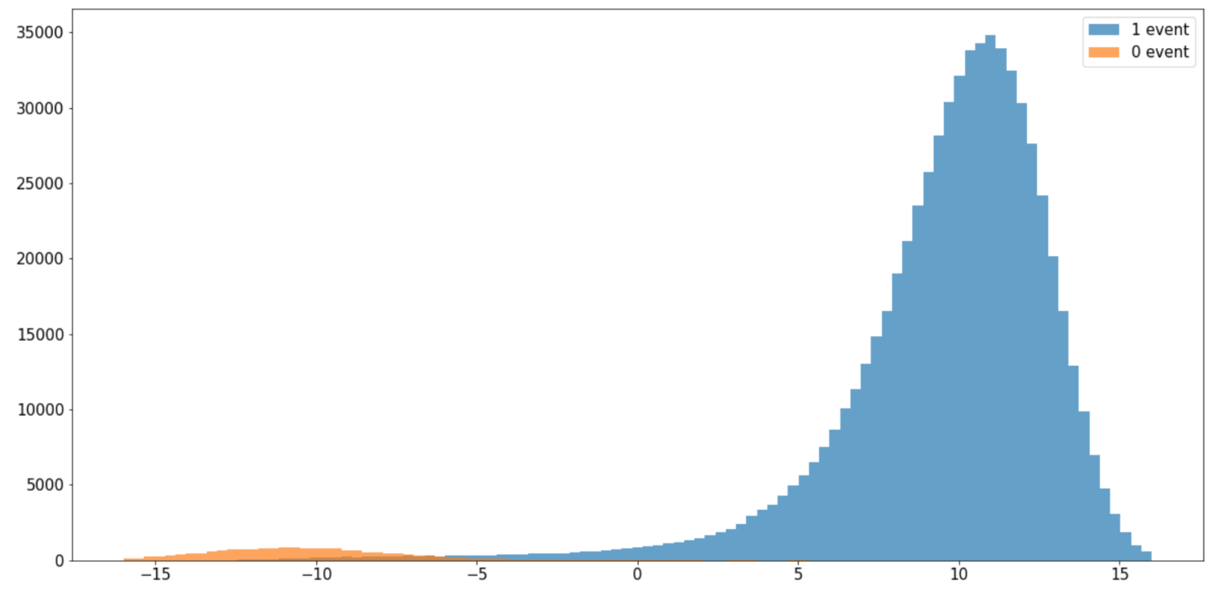
Complete training

```
=== testing results on test data ===
              precision    recall  f1-score   support

         0.0       0.60      0.97      0.74     19483
         1.0       1.00      0.98      0.99    651439

    accuracy                           0.98    670922
   macro avg       0.80      0.98      0.86    670922
weighted avg       0.99      0.98      0.98    670922

[[ 18958    525]
 [ 12777 638662]]

=== testing results on train data ===
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     60028
         1.0       1.00      1.00      1.00     60028

    accuracy                           1.00    120056
   macro avg       1.00      1.00      1.00    120056
weighted avg       1.00      1.00      1.00    120056

[[60028     0]
 [    0 60028]]
```

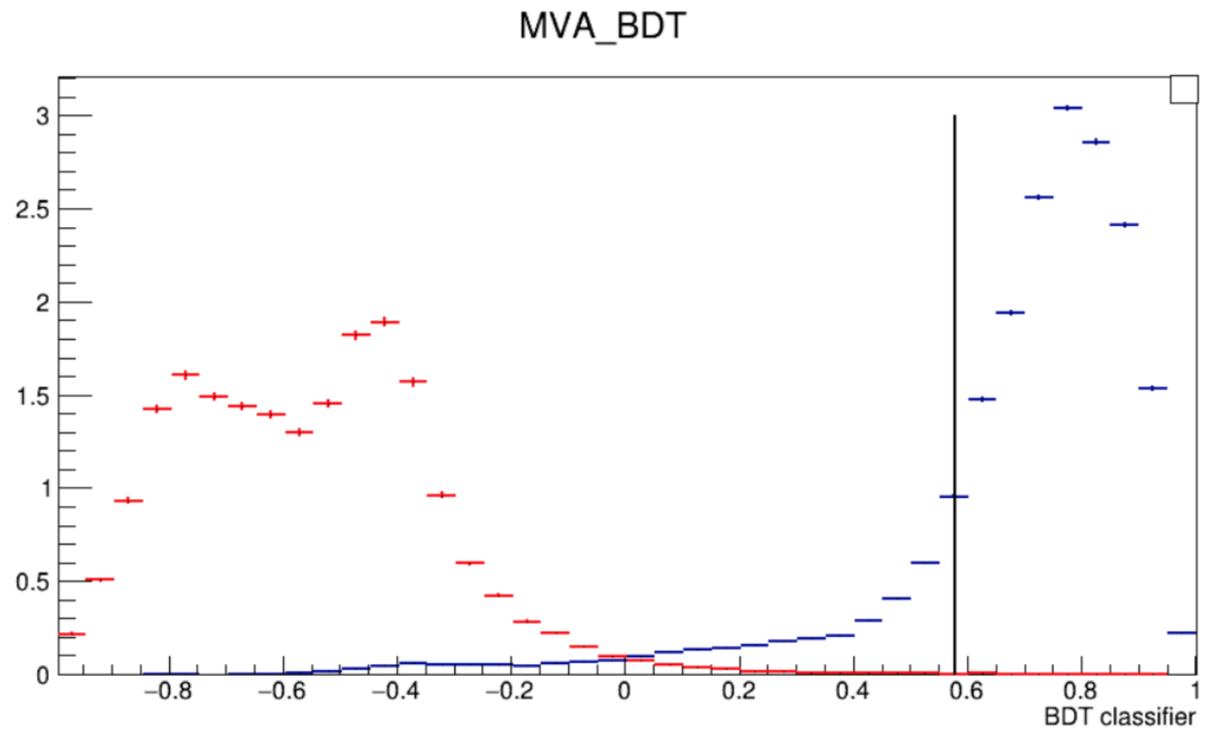
Appendix A.4: Log Scaled ROC Curve For 3 Different Models



Appendix A.5-I: Histogram Of Our Model's Raw Output



Appendix A.5-II: Histogram Of The Current Method's Raw Output



Appendix B - Neural Network Appendix

Appendix B.1: Local hit window link to examples

https://drive.google.com/file/d/1xqA65LMx7STRzEQC00QZNEo3xl1TFo7u/view?usp=sharing

Appendix B.2: Very simple neural network

```
class SimpleBinaryClassificationNet(nn.Module):
    """
    Very simple NN architecture
    """

    def __init__(self, input_size):
        super(SimpleBinaryClassificationNet, self).__init__()
        self.layer_1 = nn.Linear(input_size, 64)
        self.layer_2 = nn.Linear(64, 64)
        self.layer_out = nn.Linear(64, 1)

        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.sigmoid(x)
        x = self.relu(self.layer_2(x))
        x = self.sigmoid(x)
        x = self.layer_out(x)
        x = self.sigmoid(x)
        return x
```

Appendix B.3: Slightly more interesting neural network

```
class BinaryClassificationNet(nn.Module):
    """
    NN with batchnorms and dropout
    """

    def __init__(self, input_size):
        super().__init__()
        self.layer_1 = nn.Linear(input_size, input_size)
        self.layer_3 = nn.Linear(input_size, 250)
        self.layer_4 = nn.Linear(250, 100)
        self.layer_5 = nn.Linear(100, 30)
        self.layer_out = nn.Linear(30, 1)

        self.batchnorm2 = nn.BatchNorm1d(320)
        self.batchnorm3 = nn.BatchNorm1d(250)
        self.batchnorm4 = nn.BatchNorm1d(100)
        self.batchnorm5 = nn.BatchNorm1d(30)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        self.dropout = nn.Dropout(p=0.1)

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.batchnorm2(x)
        x = self.dropout(x)
        x = self.relu(self.layer_3(x))
        x = self.batchnorm3(x)
        x = self.relu(self.layer_4(x))
        x = self.batchnorm4(x)
        x = self.relu(self.layer_5(x))
        x = self.batchnorm5(x)
        x = self.layer_out(x)
        x = self.sigmoid(x)
        return x
```

References

Standard Model Image: *https://www.energy.gov/science/doe-explainsthe-standard-model-particle-physics*

NA62 Track Image: https://home.cern/science/experiments/na62

TRIUMF Logo: https://www.triumf.ca/home/for-media/publicationsgallery/style-guide/logo-download

UBC Logo: http://www.volleyballbc.org/post-secondary-2/ubc-logo-png-transparent/