LHCb Project – Finding $\Lambda_c^0 \to p\mu^{\pm}e^{\mp}$ Decays

Lepton flavour violation in rare, charmed baryon decay processes

Introduction

The research that we will do is similar in its methods to the PhD thesis by Dr Maciej Dudek in 2023 which can be found at https://inspirehep.net/literature/2894042 for INP Cracow. The difference between what he researched, and this project is the decay products of the Λ_c^0 particle being a muon and electron now (it was muon-antimuon in his paper). The general approach to analysis in this project will be the similar with his.

In this project ROOT and python will be used using machine learning methods (shallow neural networks) to identify signal and background. It is important to note that it is vital to not overfit with said neutral networks. There are several methods to do this that will be discussed in the theory section of this document

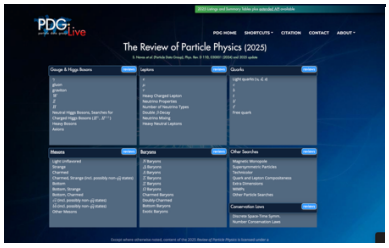
Meeting Summaries:

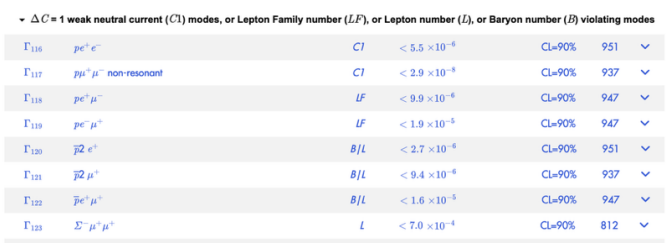
Mon 14 Jul 09:00

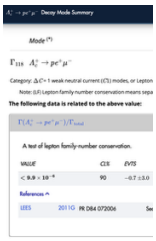
This was the first introductory meeting with Prof. Witek. We discussed all aspects of the project in rough. The strategy to analysis will involve machine learning as the amount of data recorded by the LHCb detector is so great, and as outlined in the report is reduced by about a factor of 1000 by a low-level trigger and two layers of HLT software, that result in data that is still too large. Therefore, cuts are made, and these cuts will be crucial to determine whether we can see any signal in the data.

When it comes to the tools we will use this can be separated into two components: python and root. Root will be used to initially analyse the ntuple data, which when examined with the TBrowser, can be seen to be a tree made up of hundreds of branches (different variables). These variables can be 'minimised' in a way, so that the variables are assigned a value which represents how 'good' they will be to find data – in this we mean how much distinction there is between signal and data when using the machine learning algorithms built into root. This algorithm is called TMVA and comes downloaded with the root package. It can be found by opening root as is now outlined:

Look online at PDG live for data about particles, and about the decay we are concerned with:







Tue 15 Jul 12:00

We discussed rejecting the values of the data file that were non-finite – which would break the TMVA as shown below:

```
<FATAL>                          : How am I supposed to train a NaN or +-inf?!
***> abort program execution
libc++abi: terminating due to uncaught exception of type std::runtime_error: FATAL error
```

The problem was resolved temporarily in the script code which is fully analysed in the part below. This was a different approach than that reccomended – which was to make a program that removes these number from the ntuple and allows you to have an output ntuple that can be used instead of the original that has no errors in it. This file was also created later in the day as an exercise. The problem of accesing the data on the server was fixed via a hardrive and airdropping onto my computer from Timur's. A new server that initially didn't work but as of the end of the day works was set up. The login process involves going via ssh instead of scp into a different, newly set up account.

The difference between ssh and scp:

```
SSH (Secure Shell) is for securely accessing and controlling a remote
computer, while SCP (Secure Copy Protocol) is for securely transferring
files between computers. In essence, you use SSH to get a command line
on a remote machine, and you use SCP to move files to or from it.


The login attempt is successful:
```

```
Last login: Mon Jul 14 16:17:28 on ttys004
(base) paolominhas@MacBookAir ~ % ssh lbuser9@192.245.169.53
lbuser9@192.245.169.53's password:
Linux debian10 4.19.0-18-amd64 #1 SMP Debian 4.19.208-1 (2021-09-29) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Jul 15 12:57:17 2025 from 10.10.105.106
lbuser9@debian10:~$ ls
lbuser9@debian10:~$ cd .gnupg/
lbuser9@debian10:~/.gnupg$ ls
private-keys-v1.d
```

Access was now possible however the server location was empty upon checking. This can be further discussed tomorrow. The long_list.txt was also altered removing the quotation marks and commas so that the program could read the variables.

Wed 16 Jul 10:00

We came to discuss the working server from yesterday. This means data files (ntuples) will be uploaded into this server and we will be able to access them from now on getting rid of this problem. A few major points were discussed: what to do next, the unexpected overtraining issue from running TMVA and confidence level in physics. We were given 4 papers to read, as well as some tasks relating to reducing the number of input parameters, refining the hyperparameters of BDT as well as choosing an optimal cut with the Punzi method (as outlined in one of the papers). New methods (not BDT) for the ML algorithm could be tried as well.

When the Punzi method is applied, we get ML_c oscillations (see the other papers). The data model has signal and background, and must be fit to the data:

Signal present	Signal but not enough	Signal not present
Significance > 5 sigma	3-5 sigma	Significance < 3 sigma
|	|	|
|	|	|
V	V	V
NF	‘Evidence’	‘Upper limit’ is provided
BF±	BF±	|_____|

Thu 17 Jul 09:00

Meeting was held without Timur as he was not feeling up for it (also lots of rain). The papers were to be continued to be read, as were the same tasks. In the meeting we discussed my code as it was, why there were potentially small issues where it did not look identical to his code. The final

Fri 18 Jul ----

There was no meeting as Prof. Witek had a day of leave.

Mon 21 Jul 09:00

We had a meeting in the morning to discuss the incorrect implementation of the Punzi method in my code. There was also a review of the code written in python as well as a discussion acknowledging my use of XGBoost. The test and train ROC curves were in order. The Punzi maximum point was given as 0.993 which is extremely high and unrealistic – as well as the sharp peak to the right of the graph,

whereas we expect a gentle peak in the middle or around 0.7 / 0.8 in the graph. The reason was not explored in depth but a look at checking the basics were correct concluded most likely it was fine. So, the issue remains at large.

The second part of the meeting involved Timur asking about lepton flavour violation, and whether total lepton number is conserved. It turns out that this too can be broken. Apparently, the fact total lepton number is conserved is almost a coincidence – and not a rule. Of course, from neutrino oscillation lepton flavour number is not conserved.

Tue 22 Jul 09:00

Wed 23 Jul 15:30

Note that until 14:00 we were at a lecture and tour of the IFJ facilities. Prof. Witek informed us of an issue in the input files. When aiming to increase the number of data events in the MC signal file, a cut was removed from the preselection – this resulted in the signal dataset looking different to the background dataset in a fundamental way that allowed the XGBoost to actually not look for signal, but learn how to identify the MC generated events from data from other features, leading to it heavily identifying correctly. This caused the extreme peak seen on the Punzi graph.

The data cut will be undone so that the MC signal is compatible with the background again and hopefully this will correct the issues with finding a gentle maximum on the Punzi graph.

Thu 24 Jul 12:00

Meeting to discuss application of the CLs method, as well as my issues with the graphs. The issues were most likely due to a few factors: adjustment of hyperparameters as there is significant overtraining, the addition of a cut of all data below 2300 on the graph to get rid of backgrounds that originate from collisions that give a large signature, and the use of an incorrect datafile. The data file has been updated so the new one needs to be downloaded. It is with this that the correct Punzi graph will come.

Daily Summary of Work Achieved

Mon 14 Jul

Trying to open the ntuple file using scp was to no avail as

Tue 15 Jul

ROOT TMVA Macro

A file was written in C++ to use the TMVA ROOT tool. The file is outlined below:

The first step is to import the relevant libraries. We used standard ROOT libraries of TFile, Ttree, TBranch, TObjArray & some TMVA libraries TMVA/Factory, TMVA/Tools. Note that <iostream> was not included as in a root macro it is not necessary to have a main function and therefore to use standard C++ conventions, as the file is never formally compiled (unless you choose to do this) so the main function is commented out. It is however very important that the function used in the file has the same name as that of the file, otherwise the file will not work.

The function is defined as: `void TMVA_test() {`

The TMVA library is loaded: `TMVA::Tools::Instance();`

A ROOT file is opened to store output: `TFile *outputFile = TFile::Open("TMVAOutput.root", "RECREATE");`

A TMVA factory is defined with some conditions:

```
TMVA::Factory *factory = new TMVA::Factory("TMVAFactory", outputFile, "!V:!Silent:Color:DrawProgressBar:AnalysisType=Classification");
```

The dataset is initialised: `TMVA::DataLoader *dataloader = new TMVA::DataLoader("dataset");`

Now the two input files, the MC and the background data are taken as input and the trees taken in and defined as signal and background trees:

```
TFile *inputFileSig = TFile::Open("Lc2pemu_MC.root");
TFile *inputFileBkg = TFile::Open("Lc2pemu_DATA_osign_noBrem.root");
if (!inputFileSig || inputFileSig->IsZombie()) { std::cout << "Error:
if (!inputFileBkg || inputFileBkg->IsZombie()) { std::cout << "Error:


TTree *signalTree = (TTree*)inputFileSig->Get("DecayTree");
TTree *backgroundTree = (TTree*)inputFileBkg->Get("DecayTree");
```

Note that there are two checks to see if the input files were opened correctly and an error is printed if they were not.

After this the list of variable names that are going to be trained on is opened and changed to a vector of the variable names. The code skips blank lines in the file and also checks that the file was opened correctly:

```
// --- Read variable names from the text file ---
std::vector<TString> trainingVars;
std::string varName;
std::ifstream varFile("long_list.txt"); // <-- Open text file

if (!varFile.is_open()) {
    std::cerr << "Error: Could not open variables file!" << std::endl;
    return;
}

while (std::getline(varFile, varName)) {
    if (!varName.empty()) { // Avoid adding empty lines
        trainingVars.push_back(varName.c_str());
    }
}
varFile.close();
// ------------------------------------------------

// Now, add the variables from the vector to the dataloader
std::cout << "--> Adding the following training variables: " << std::endl;
for (const auto& var : trainingVars) {
    std::cout << "  - " << var << std::endl;
    dataloader->AddVariable(var, 'F'); // 'F' for Float
}
```

Now the trees are loaded with the. Factory. This means the training data is being prepared as well as the testing data. The weight of each event is assigned to be 1.0:

```
Double_t signalWeight     = 1.0;
Double_t backgroundWeight = 1.0;
dataloader->AddSignalTree(signalTree, signalWeight);
dataloader->AddBackgroundTree(backgroundTree, backgroundWeight);
```

There is now one cut. This removes any events that have non-finite values that will break the TMVA machine.

```
// TMath::Finite() returns true if the number is not NaN or infinity.
TCut mycuts = ""; // Initialize with no cuts
for (const auto& var : trainingVars) {
    mycuts += TCut(TString::Format("TMath::Finite(%s)", var.Data()));
}
```

Now we book methods for classification. BDT is used as it is good for variable importance (apparently) however tomorrow it would be good to try a few different methods.

```
dataloader->PrepareTrainingAndTestTree(mycuts, "nTrain_Signal=0:nTrain_Background=0:SplitMode=Random:NormMode=NumEvents:!V");
factory->BookMethod(dataloader, TMVA::Types::kBDT, "BDT", "!H:!V:NTrees=400:MinNodeSize=2.5%:MaxDepth=3:BoostType=AdaBoost");
```

Finally, all methods are trained, then tested then evaluated. This is a standard block for using TMVA:

```
factory->TrainAllMethods();
factory->TestAllMethods();
factory->EvaluateAllMethods();
```

The output file is now closed and the name of the output file is printed (so I do not forget what file to run!)

```
outputFile->Close();
std::cout << "==> Wrote root file: " << outputFile->GetName() << std::endl;
```

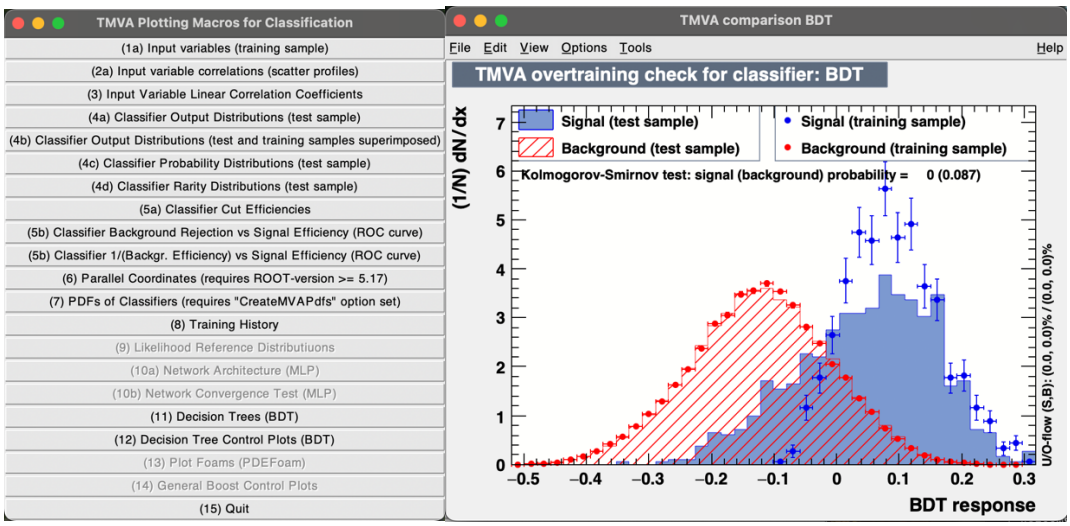
Note that the factory is not manually closed at the end of the code. It is best to let root handle this automatically, as manually forcing it can cause errors at times. It is now important to define how the file is first run to produce the output root file, and then how the output file is run in ROOT to see the GUI. The file is run in the terminal as following:

```
(base) paolominhas@MacBookAir ~ % root -l -q "TMVA_test.C"
```

The -l runs root without the startup splash across the screen, and the -q tells root to run the file in quotation marks then immediately close root. This allows the file to be run in root inside the terminal without opening the root environment. When run, the file outputs progress bars (as defined in the code above), telling you the processes happening during training and then testing. The output file is then accessed in the root environment by looking at it with the TMVA Gui:

```
(base) paolominhas@MacBookAir ~ % root -l

root [0] TMVA::TMVAGui("TMVAOutput.root")
```

This loads the gui interface where the ML results can be analysed:



Wed 16 Jul

The first task undertaken in the morning was a program to transfer data from the ROOT ntuple format to a format that could be used with pandas in python, to allow for a different type of ML analysis. There are two approaches outlined online. The first is to install “uproot” in python and then just use this package (which doesn’t mean you need to even have root installed on your computer). The other approach is to do it directly with root. This is slightly more tedious but is the approach I have chosen as we are also using C++ ROOT so this is better for our applications.

The first step is to “source root”:

```
(base) paolominhas@MacBookAir ~ % source /opt/homebrew/bin/thisroot.sh
```

The file would not run as there were conflicting versions of python. I installed a new conda environment but still disagreements between the version of python ROOT could work with

Thu 17 Jul

First the implementation of the same method in python was carried out. It was necessary, due to the clash of different python versions, to simplify running of the file in python by using uproot instead of native ROOT. This meant a slightly different implementation of the file transfer (it was still very simply two lines using pandas to change the ntuple to a dataframe).

```
import uproot

import pandas as pd



with uproot.open("Lc2pemu_MC.root") as file:

    tree = file["DecayTree"]

    df = tree.arrays(library="pd")

# This is for the signal file
```

The file was then altered from this basic transfer, into a python file to perform the same analysis as TMVA, however using scikit learn. Firstly, data was loaded from files:

```
file_sig = "Lc2pemu_MC.root"
file_bkg = "Lc2pemu_DATA_osign_noBrem.root"
tree_name = "DecayTree"
features = np.loadtxt("long_list.txt", dtype=str).tolist()
```

In the list of features, an issue arose as one variable was mentioned twice so one of the instances was deleted:

```
13    Lc_IPCHI2
14    Lc_VTXCHI2DOF
15    p_OWNPV_CHI2
16    p_IP_OWNPV
17    p_IPCHI2_OWNPV
18    p_IPCHI2_OWNPV
19    p_PT
20    p_uniProbNNp
21    DTF_CHI2
22    p_FITPV_IPCHI2
23    FITPV_CHI2
```

Now the ntuples were converted into pandas dataframes:

```
with uproot.open(file_sig) as f:
    df_sig = f[tree_name].arrays(features, library="pd")
with uproot.open(file_bkg) as f:
    df_bkg = f[tree_name].arrays(features, library="pd")
```

The signal and background dataframes are then combined to create a single dataset and an extra column is added with values of 1 for signal and 0 for background:

```
df_sig['target'] = 1    # Signal events labeled as 1
df_bkg['target'] = 0    # Background events labeled as 0
df = pd.concat([df_sig, df_bkg], ignore_index=True)
print(f"Data loaded. Total events: {len(df)}, Signal: {len(df_sig)}, Background: {len(df_bkg)}")
```

A cut is made as in TMVA to get rid of the values of infinity by changing them to NaN and then using the pandas library to remove all NaN values from the dataset:

```
print(f"Original number of events: {len(df)}")
# Replace infinite values with NaN (Not a Number)
df.replace([np.inf, -np.inf], np.nan, inplace=True)
df.dropna(inplace=True)
print(f"Number of events after cleaning: {len(df)}")
```

Then the feature matrix (the features that are used for training) and target vector (the list of 0s and 1s for background and signal) are defined as dataframes:

```
X = df[features]
y = df['target']
```

The data is split into 70% train and 30% test – which is the same way TMVA performs analysis.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

print(f"Training set size: {len(X_train)}, Testing set size: {len(X_test)}")
```

The training is then performed using the GradientBoostingClassifier. As described on scikit learn: Gradient Boosting for classification. “This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage `n_classes_` regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only a single regression tree is induced”. This is the most similar choice to TMVA in ROOT so was used for this reason.

```
bdt = GradientBoostingClassifier(
    n_estimators=400,     # Number of trees (NTrees in TMVA)
    max_depth=3,          # Max depth of each tree
    learning_rate=0.1,    # Learning rate (Shrinkage in TMVA)
    subsample=0.5,        # Fraction of samples used for fitting each base learner
    random_state=42       # Random state for reproducibility
)
print("\nTraining the BDT model...")
bdt.fit(X_train, y_train)
print("Training complete.")
```

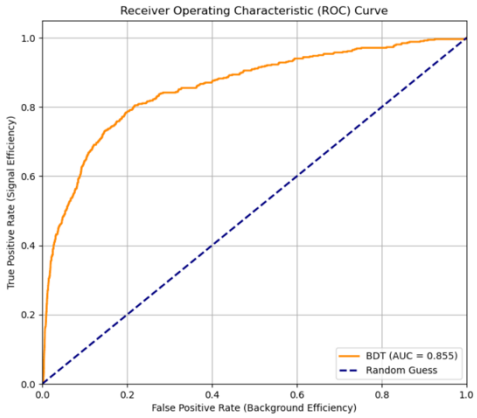
Now the data is evaluated by getting the neural network to predict whether an event is signal or data by returning a 0 or a 1. This is then compared to the real values:

```
y_pred_proba = bdt.predict_proba(X_test)[:, 1]
print("\nClassification Report on Test Set:")
y_pred_class = bdt.predict(X_test)      # predict() gives 0 or 1, predict_proba() gives the probability
print(classification_report(y_test, y_pred_class, target_names=['Background', 'Signal']))
```

Now the area under the curve and the line on the ROC curve are calculated and plotted by GBC. This is plotted with matplotlib:

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

    roc_auc = auc(fpr, tpr)



    plt.figure(figsize=(8, 8))

    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'BDT (AUC =
{roc_auc:.3f})')

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random
Guess')

    plt.xlim([0.0, 1.0])

    plt.ylim([0.0, 1.05])

    plt.xlabel('False Positive Rate (Background Efficiency)')

    plt.ylabel('True Positive Rate (Signal Efficiency)')

    plt.title('Receiver Operating Characteristic (ROC) Curve')

    plt.legend(loc="lower right")

    plt.grid(True)

    plt.show()
```

Returning a ROC graph as follows:



The fit here is not as good as the one presented by professor Witek – he had used a different algorithm from scikit learn – it is worth finding out what this one is and then also implementing it. After this section was written the code was written up in OOP style – it would be nice to make classes for the neural network and implement it properly as Lisa had in the 3 lectures in the morning this week.

The next part of the day was spent reading and understanding the papers provided. These will now be summarised below:

Papers today:

Sensitivity of searches for new signals and its optimisation – Giovanni Punzi

The definition of 'confidence level' had not been clearly defined in particle physics in 2003, and Giovanni Punzi wrote this paper to clearly define the term. The terms defined by characterising the 'sensitivity'. Or at least the sensitivity region

Fri 18 Jul

The same tasks from the previous day were completed and attempted. See Mon 21 Jul to see the full development.

Mon 21 Jul

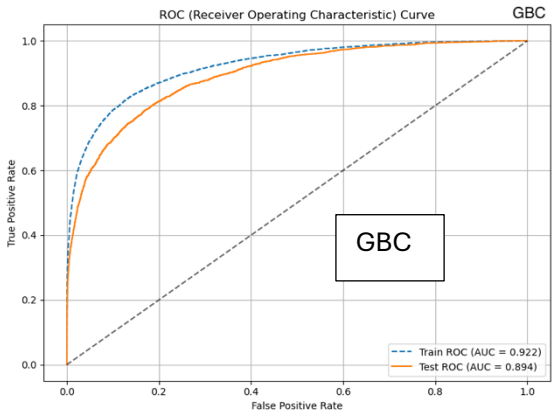
The code was greatly developed between Friday and Monday. The python program was split into several files to be more ordered. The Plotting.py file has the relevant functions for plotting different graphs and printing a pdf file as output with these graphs. The Statistics.py file includes the Punzi calculation and the root ntuple file output function. The major changes are as follows:

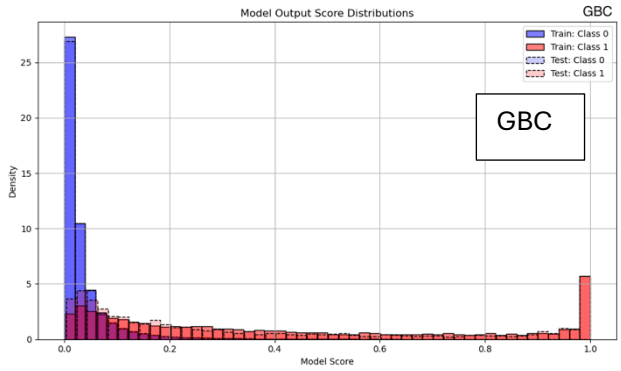
- Output of a root nutuple in which is the combined signal and background data with the machine learning calculated probability values of signal and data added as a final column
- Calculation of a Punzi score and plotting to see where the maximum lies
- New graphing with better colours
- Outputting a pdf file with the graphs on
- The implementation of a new method from GBC to XGB
- There are therefore two methods of machine learning – now only XGB is being used

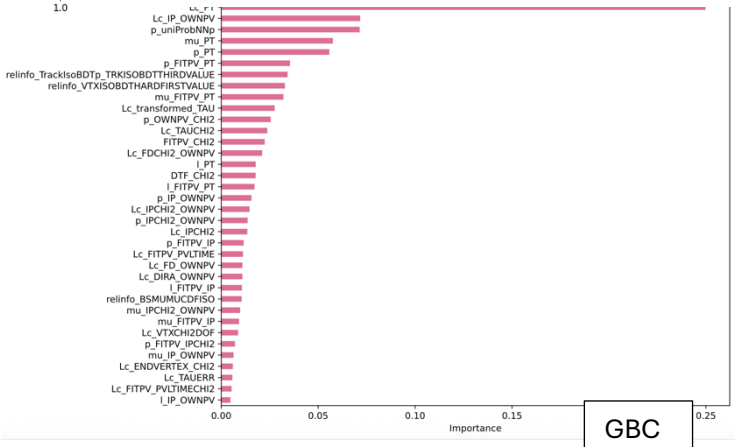
The implementation of XGB involved using some selection. This reduced the number of parameters we were probing to 18 by picking the most useful.

```
Function for selection:

prelim_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)

prelim_model.fit(X_train, y_train)  # Train preliminary XGBoost model on all features

selection = SelectFromModel(prelim_model, threshold='median', prefit=True)

    # pick features with importance above the median

selected_features = X_train.columns[(selection.get_support())]

selected_feature_names = X_train.columns[(selection.get_support())]

print(f"Selected {len(selected_features)} features out of {len(X_train.columns)}.")

X_train_selected = selection.transform(X_train)

X_test_selected = selection.transform(X_test) # Transform data keep selected features

print("Selected features:", selected_features.tolist())
```

Below we can see the results from the initial implementation of the GBC method:

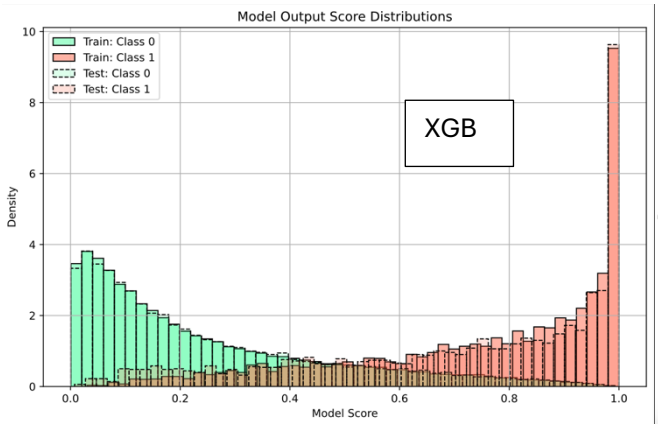


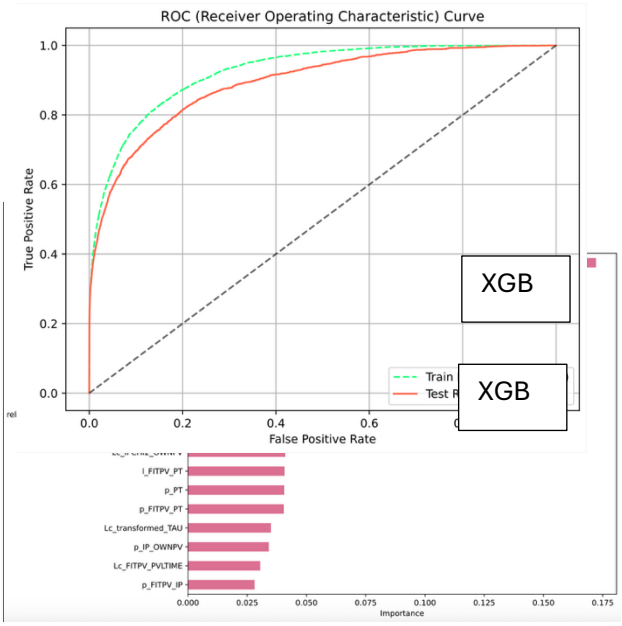




Note the very high peaks for background assigned a value of 0 and how low this is in other places. The method is quite extreme in assignment.

There are also many hyper parameters which have low importance as can be seen in the third figure, that are being considered. It would be best to reduce this as was done with the XGB. The ROC curve works better than before, with a training value of over 0.9 and testing just under 0.9. This provides a relatively high area under the curve. The testing value is naturally lower than that of the training, as it is less efficient.

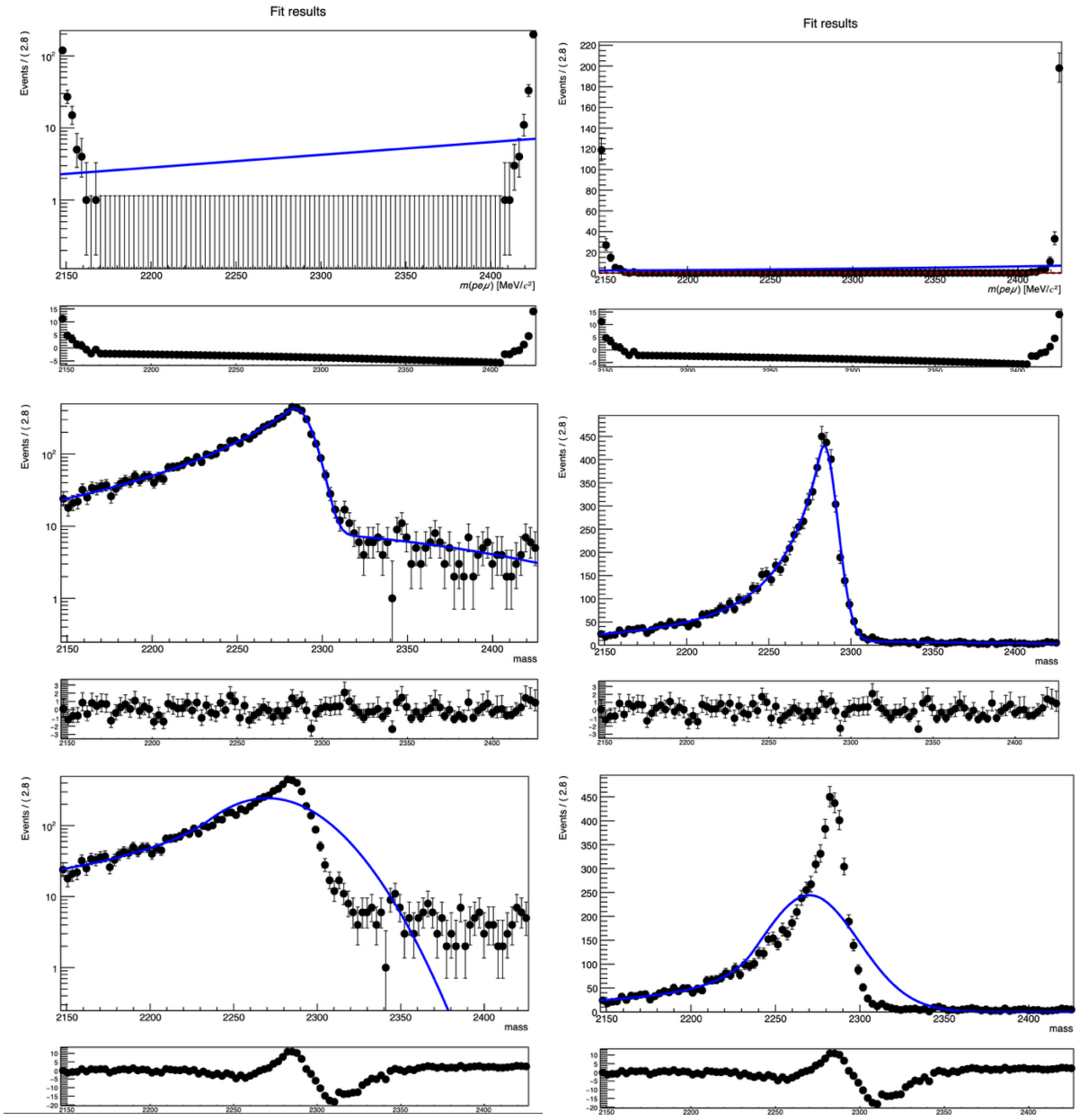




For XGB we can see that there is a greater spread of identification values – especially those for background. The classification of values must therefore be more nuanced as before values were being assigned heavily values close to one and zero.

Tue 22 Jul

Prof. Witek sent a new file which was downloaded and implemented to use rufit to analyse the output data in the morning. Upon changing input file directories, the code was run giving the following plots (some of which did not work as planned):



From left to right top row: mass (log), mass. Then the next four graphs are signal shape (crystal ball): gauss, gauss log, lod and normal.

```
(base) paolominhas@MacBookAir Run2 % scp -r Run2  lbuser9@192.245.169.53:/home/lbuser9/studentupload

scp: stat local "Run2": No such file or directory
(base) paolominhas@MacBookAir Run2 % cd ..
(base) paolominhas@MacBookAir OutputData % scp -r Run2  lbuser9@192.245.169.53:/home/lbuser9/student
upload
lbuser9@192.245.169.53's password:
.DS_Store                                     100% 6148     1.0MB/s   00:00
XGBoostPlots.pdf                              100%   49KB   2.5MB/s   00:00
bdt_output_preliminary.root                   100%  290MB   2.3MB/s   02:07
```

The meeting was had from

Papers Today

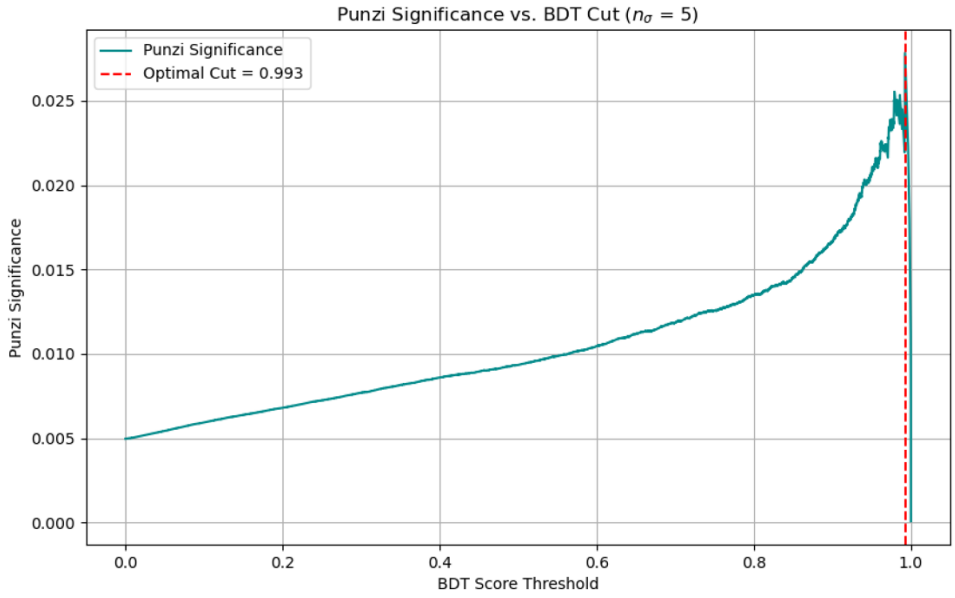
A practical guide to statistics techniques in particle physics

Modified frequentist analysis of search results (the CLs method)

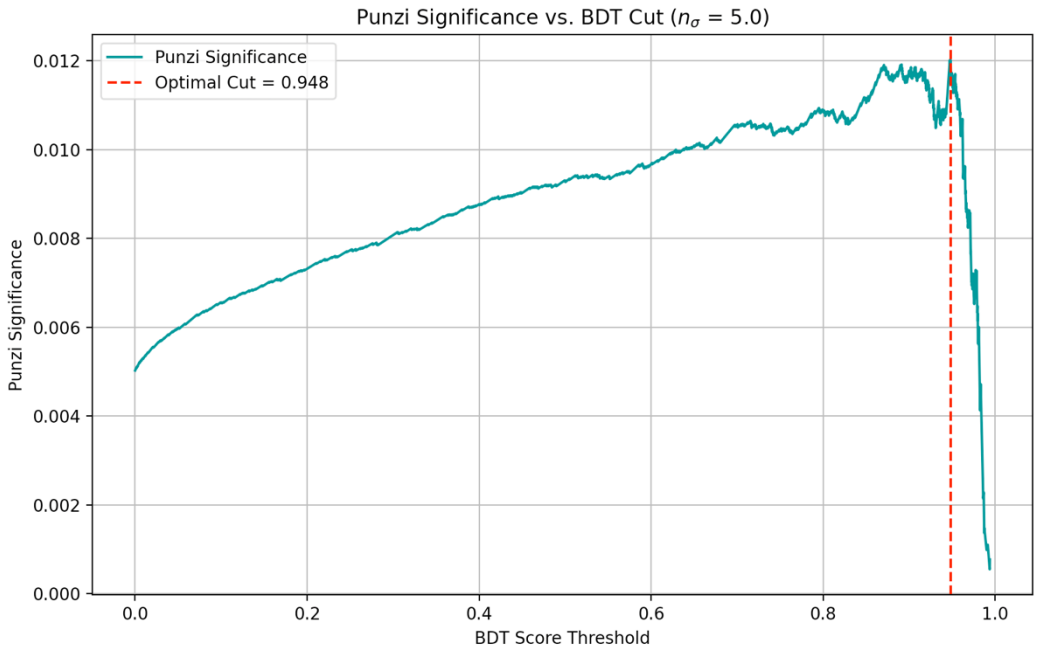
Presentation of search results: the CL s technique

Wed 23 Jul

Currently the Punzi graph shows an extremely sharp peak around 0.993:



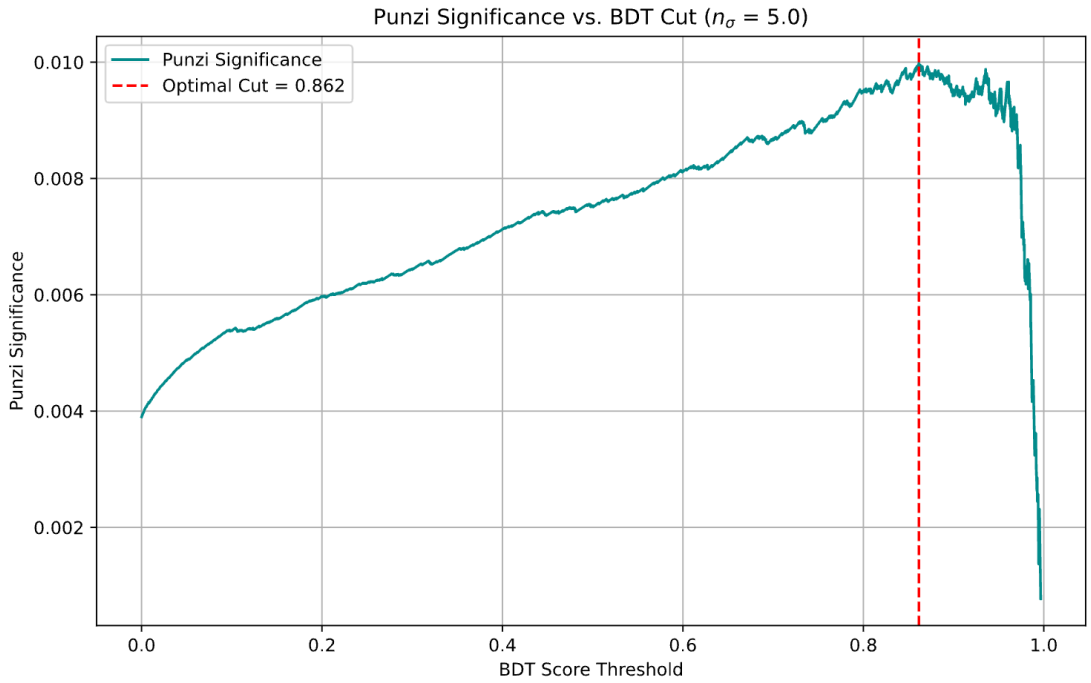
This graph does not appear correct. Efforts to correct this were attempted through changing the Punzi formula in various ways and examining the efficiency values I obtained. The real problem mas the MC data. The corrected signal input file was downloaded and the old file renamed to Lc2pemu_MC_old.root.



This is the new graph, which shows some improvement but not much.

Thu 24 Jul

The issue with the Punzi graph was investigated. The new data was used which resulted in the following graph:



Fri 25 Jul

In order to determine the differences between my results and Prof. Witek's, it is necessary to compare the code when the model is being trained.
My code:

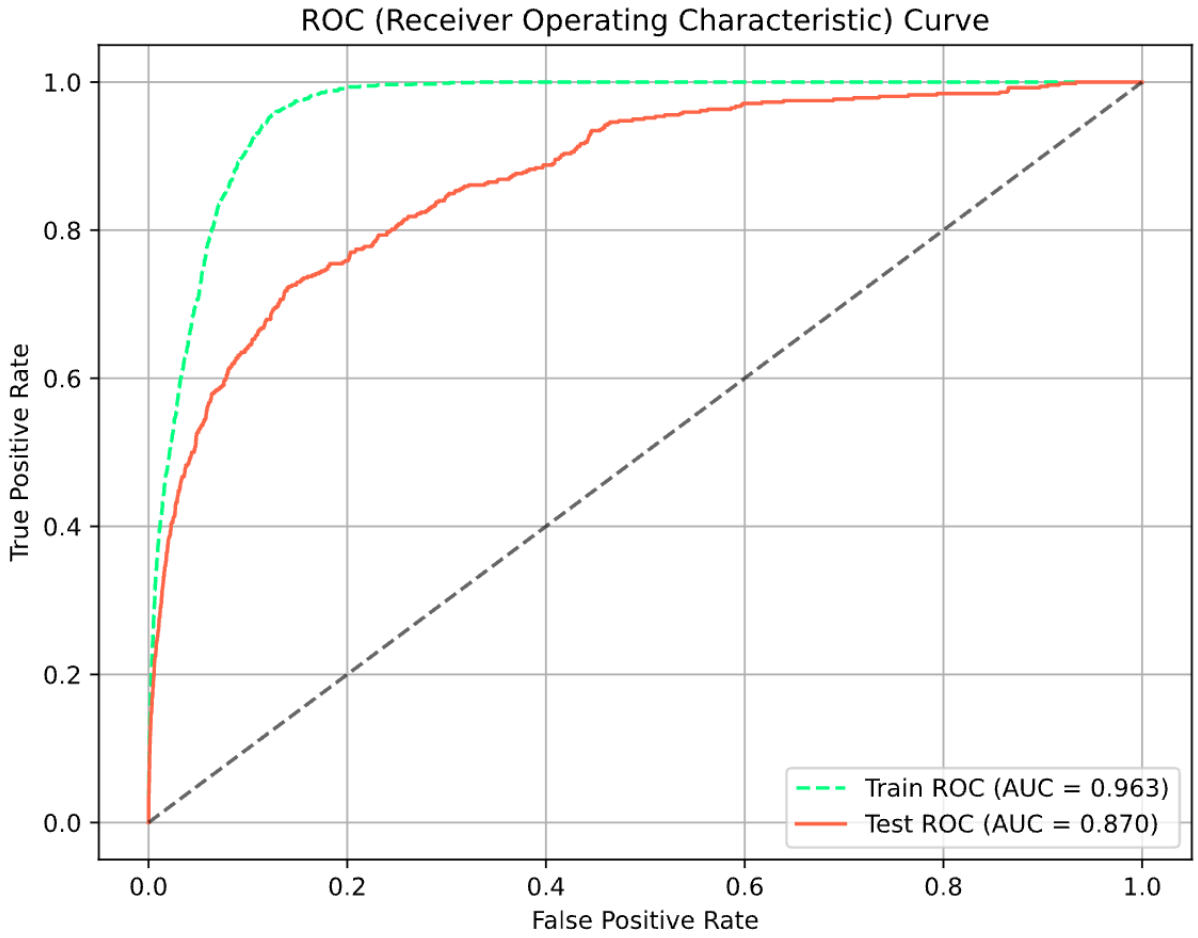
```
print("\nPerforming feature selection...")

    prelim_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
random_state=42)

    prelim_model.fit(X_train, y_train)  # Train preliminary XGBoost model on all
features

    selection = SelectFromModel(prelim_model, threshold='median', prefit=True) #
pick features with importance above the median



    selected_features = X_train.columns[(selection.get_support())]

    selected_feature_names = X_train.columns[(selection.get_support())]

    print(f"Selected {len(selected_features)} features out of
{len(X_train.columns)}.")

    X_train_selected = selection.transform(X_train)

    X_test_selected = selection.transform(X_test) # Transform data keep only
selected features

    print("Selected features:", selected_features.tolist())
```

```
    ########################################################
    #                                                      #
    #               Training & Evaluation                  #
    #                                                      #
    ########################################################


    final_bdt = XGBClassifier(
        n_estimators=400,                                          # Number of
trees (NTrees in TMVA)
        max_depth=3,                                               # Max depth
of each tree
        learning_rate=0.1,                                         # Learning
rate (Shrinkage in TMVA)
        subsample=0.5,                                             # Fraction
of samples used for fitting each base learner
        scale_pos_weight=np.sum(y_train == 0) / np.sum(y_train == 1),   # Handle
class imbalance
        use_label_encoder=False,                                   # Avoid
warning about label encoder
        eval_metric='logloss',                                     #
Evaluation metric
        random_state=42                                            # Random
state for reproducibility
    )
    final_bdt.fit(X_train_selected, y_train)
```

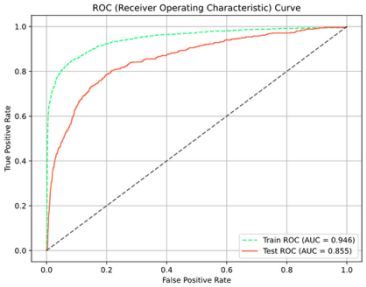
Prof. Witek's code:

```
tag=[1]*n_S+[0]*n_B
    df_y=pd.Series(tag, name='tag')
    df_X=pd.concat([df_S,df_B])
    print("S+B dataframe size ", df_X.shape[0])
    print(df_X.head())

    # Step 5: Train/Test and XGBoost model with feature selection
    X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.4,
random_state=42)

    # Initial model
    model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
max_depth=3 )
    model.fit(X_train, y_train)
```

The major issue with my code is that it is certainly grossly overtraining, as is clear by the XGB graph here:



Even compared to GBC below which is a different method, uses less hyperparameters and most crucially doesn't make an additional test model to select parameters then make a main model as I have done in the code. I think it can be safe to conclude it is this which is having a large impact on the training so this part will be rewritten in a fully new directory and package.



To outline what I have done originally, we can select the part of code which was removed – principally the automated data selection section:

```
print("\nPerforming feature selection...")
prelim_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss',
random_state=42)
```

The first section creates a preliminary model with XGB with standard hyperparameters (use_label_encoder is set to false by default and means the data is used exactly as provided without changing the target variable values (y) into integers (like 0, 1, 2, 3 ...), and logloss (this heavily penalises confident and wrong selections so is used for classification problems – it aims to improve the model genuinely between runs). The random state of 42 is chosen as 42 has no specific special mathematical properties – this state is used for reproducibility – so that the same order of randomness could be made by someone else.

```
prelim_model.fit(X_train, y_train)
selection = SelectFromModel(prelim_model, threshold='median', prefit=True)
```

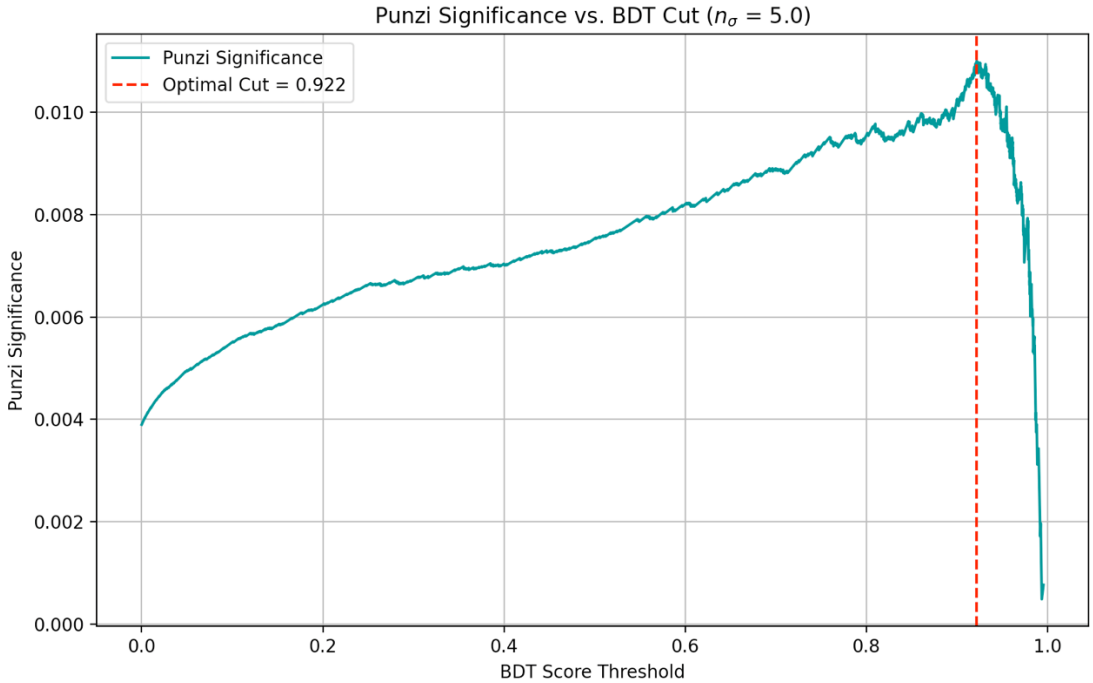
The code now fits the model and makes the selection of 'best' variables. The scikit learn function SelectFromModel removes features based on an impotance level. It uses the prelim_model which already knows which features are most useful for making predictions. The rule for which features to keep is set by the median threshold. This calculates the importance of the features and then finds the median value – then selects to keep only the values above the median values thus cutting off 50% of the features. This results in 18 features being kept. Prefit=true is just telling the program that prelim_model has already been trained so it doesn't train it again.

```
selected_features = X_train.columns[(selection.get_support())]
selected_feature_names = X_train.columns[(selection.get_support())]
print(f"Selected {len(selected_features)} features out of {len(X_train.columns)}.")
```

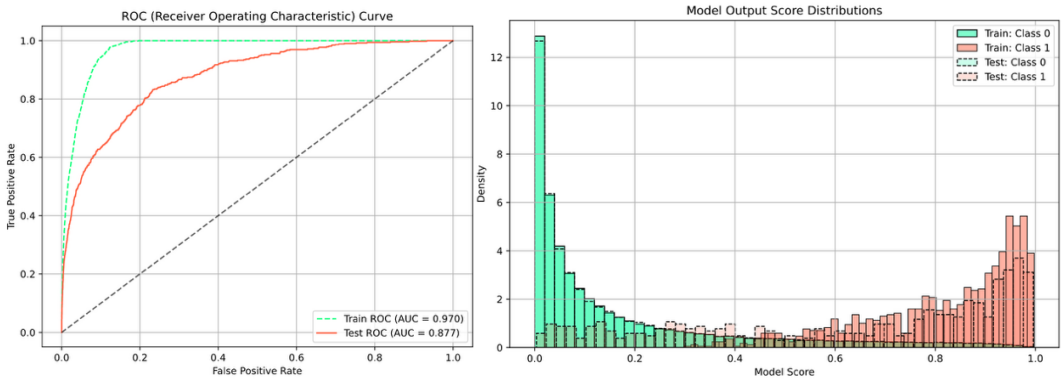
selected features takes only the features of X_train that were slceted y the model (as selection_get_support() returns Boolean array where values are Ture for features kept or else false).

```
X_train_selected = selection.transform(X_train)
X_test_selected = selection.transform(X_test)
print("Selected features:", selected_features.tolist())
```

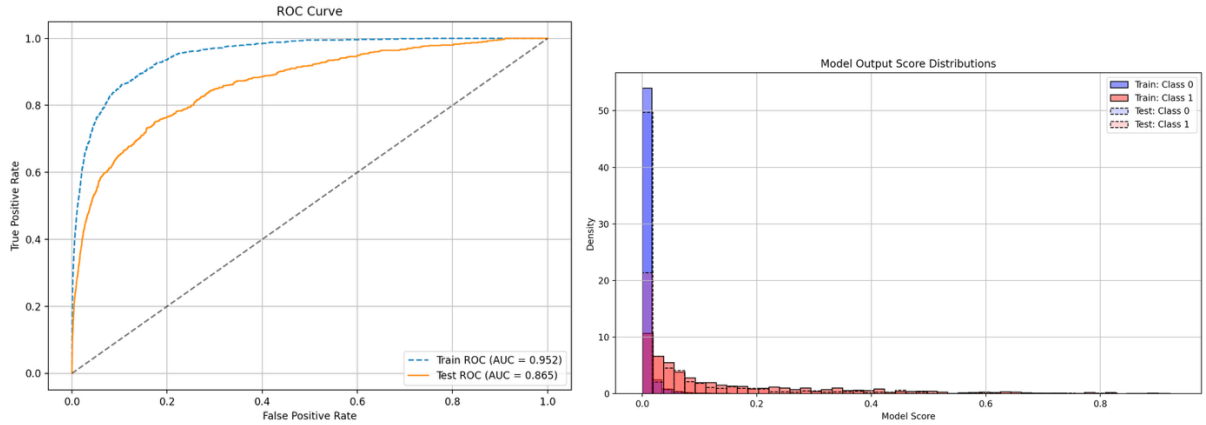
The 'unimportant' features are removed from the dataframe and thus the selected data is used here. This most likely causes the overfitting – it will now be tried with out this part and just using the original input data. Upon removing this the Punzi graph appeared as follows:



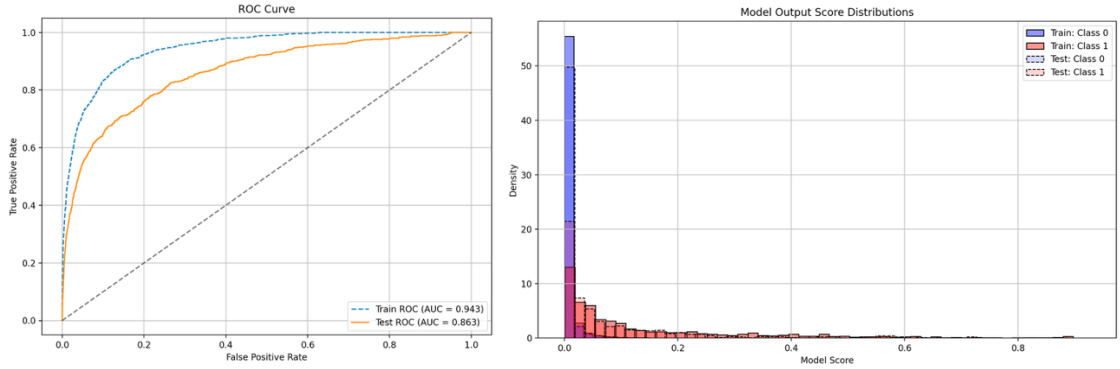
This is arguably worse than the previous graph. The overtraining aspect can be determined from the other graphs which are displayed below. It is now inherently clear that the graphs show even more significant overtraining. Therefore it is necessary to tune other aspects of the code clearly.



Upon further inspection in fact both code files have the preliminary model approach so the section explained above was readded back in the file



This graph is from the method implemented by Prof. Witek showing a lower training AUC.



And for the second, selected model the results are marginally more optimal.

Zfit program:
There were again issues with dependencies clashing. It was recommended to create a conda environment:

```
(base) paolominhas@MacBookAir PythonScripts % conda create -n hep-analysis python=3.9
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /Users/paolominhas/opt/anaconda3/envs/hep-analysis

  added / updated specs:
    - python=3.9


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    openssl-3.0.17             |       hee2dfae_0         4.6 MB
    python-3.9.23              |       hd8516d5_0        12.6 MB
    setuptools-78.1.1          |   py39hecd8cb5_0         1.6 MB
    sqlite-3.50.2              |       hc8b0dd6_1         1.1 MB
    wheel-0.45.1               |   py39hecd8cb5_0         117 KB
    ------------------------------------------------------------
                                           Total:        20.0 MB

The following NEW packages will be INSTALLED:

  bzip2              pkgs/main/osx-64::bzip2-1.0.8-h6c40b1e_6
  ca-certificates    pkgs/main/osx-64::ca-certificates-2025.2.25-hecd8cb5_0
  expat              pkgs/main/osx-64::expat-2.7.1-h6d0c2b6_0
  libcxx             pkgs/main/osx-64::libcxx-17.0.6-hf547dac_4
  libffi             pkgs/main/osx-64::libffi-3.4.4-hecd8cb5_1
  ncurses            pkgs/main/osx-64::ncurses-6.4-hcec6c5f_0
```

Then upon activating said environment:

```
(base) paolominhas@MacBookAir PythonScripts % conda activate hep-analysis
(hep-analysis) paolominhas@MacBookAir PythonScripts %
```

Then the relevant packages for using the zfit functions were installed inside this environment: `conda install -c conda-forge zfit mplhep` which was then extended to include more packages:

```
(hep-analysis) paolominhas@MacBookAir PythonScripts % conda install -c conda-forge zfit mplhep numpy uproot matplotlib Path
Collecting package metadata (current_repodata.json): \ WARNING conda.models.version:get_matcher(544): Using .* with relation
ure version of conda. Your spec was 1.7.1.*, but conda is ignoring the .* and treating it as 1.7.1
done
Solving environment: failed with initial frozen solve. Retrying with flexible solve.
Solving environment: failed with repodata from current_repodata.json, will retry with next repodata source.
Collecting package metadata (repodata.json): / WARNING conda.models.version:get_matcher(544): Using .* with relational opera
ion of conda. Your spec was 1.8.0.*, but conda is ignoring the .* and treating it as 1.8.0
WARNING conda.models.version:get_matcher(544): Using .* with relational operator is superfluous and deprecated and will be r
is ignoring the .* and treating it as 1.9.0
WARNING conda.models.version:get_matcher(544): Using .* with relational operator is superfluous and deprecated and will be r
is ignoring the .* and treating it as 1.6.0
done
Solving environment: -
```

This did not end up working and was temporarily abandoned.

Mon 28 Jul

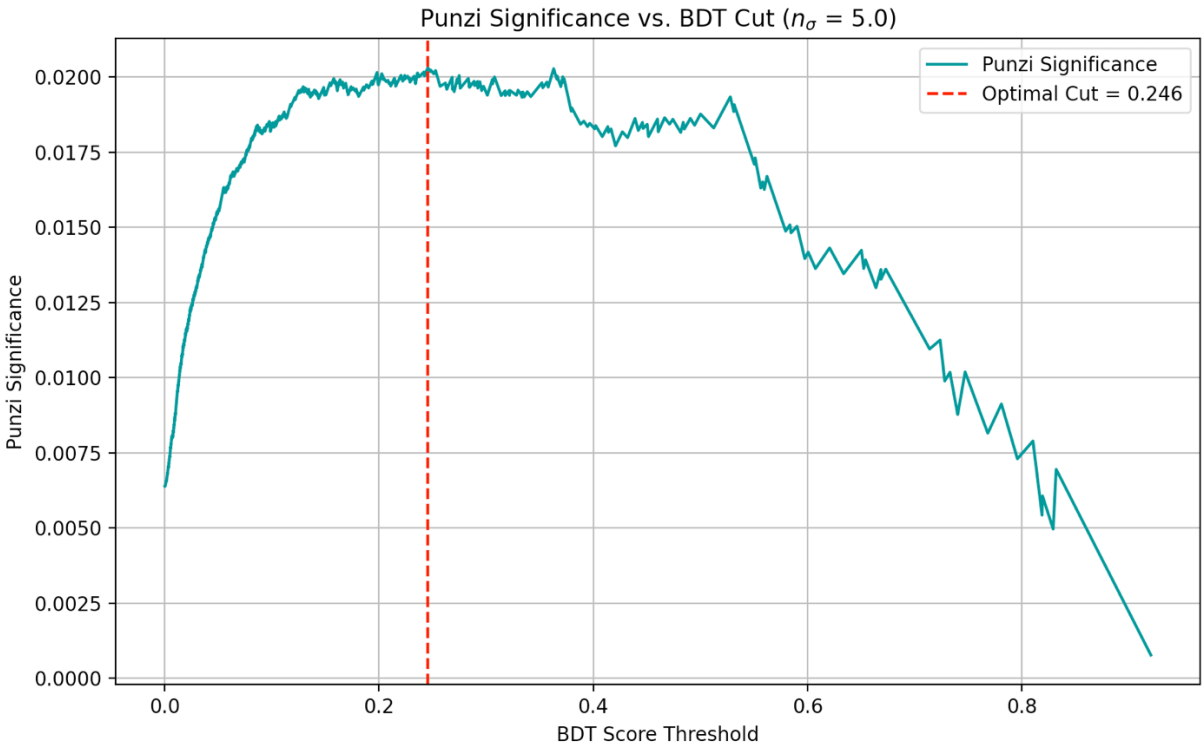
The main issue to fix today (and over the previous weekend) is the overtraining of my model. It now becomes necessary to compare it directly to Prof. Witek's model, as his is a more simple one and this could be the required strategy.

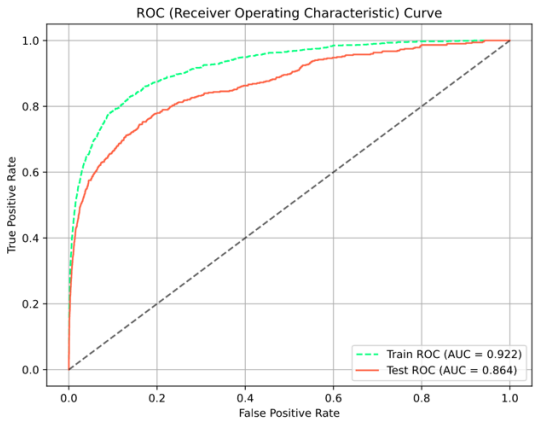
The number of estimators (this is a hyperparameter) in my model is higher than that of his – meaning the model Is more complex. There are are other small issues. I still need to add the cut above 2300 but this has little effect and can be done easily. The hyperparameters used are different. For example, Prof. Witek uses 100 estimators (I use 400), a min child weight of 1 (I use 10, meaning smaller more noisy groups could be included), a subsample of 1 .0 (I used 0.5 to reduce overfitting

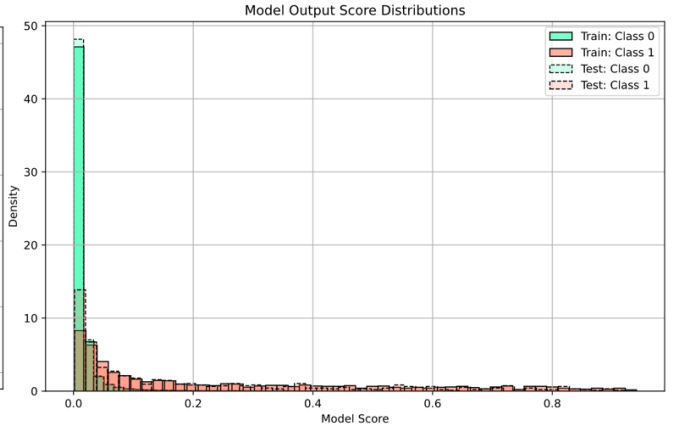
but likely has little impact) and did not use scale_pos_weight (I weighted the classes to remove an imbalance which hopefully proves useful – it does not increase overfitting).

I also split the data differently – we both do a 70-30 train-test split but I uses stratify=y which ensures the test and train samples have the same proportions of signal and background data. However, the biggest difference between models is in the feature selection – currently I had removed the selection and have forgotten to re-add it back in so will do this now. It turns out that the overtraining was caused by the hyperparameters. Once changing to the following the Punzi graph appeared as expected:

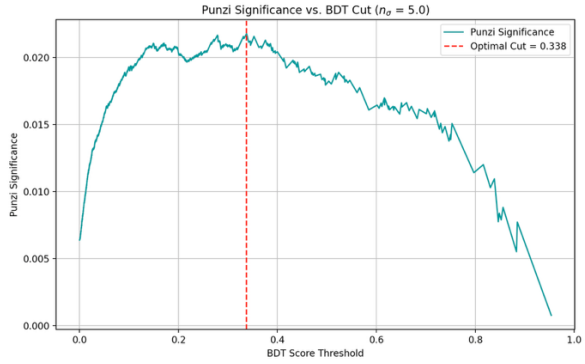
```
final_bdt = XGBClassifier(
    n_estimators=100,                                    # Number of trees (NTrees in TMVA)
    max_depth=3,                                         # Max depth of each tree
    learning_rate=0.1,                                   # Learning rate (Shrinkage in TMVA)
    #subsample=0.5,                                       # Fraction of samples used for fitting each base learner
    #scale_pos_weight=np.sum(y_train == 0) / np.sum(y_train == 1),   # Handle class imbalance
    min_child_rate=10,                                  # Minimum child weight (min_child_weight in TMVA)
    use_label_encoder=False,                             # Avoid warning about label encoder
    eval_metric='logloss',                               # Evaluation metric
    random_state=42                                      # Random state for reproducibility
)
```

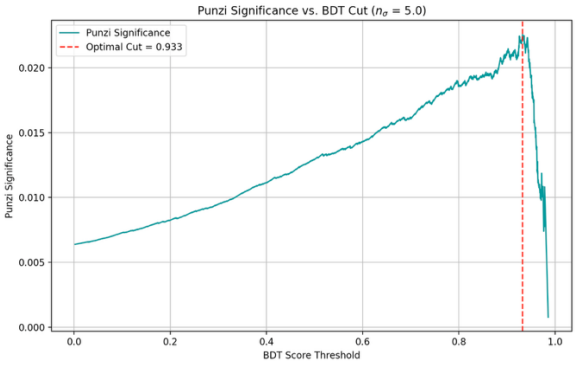






Now that the issues with the model have been resolved it is worth probing deeper to the specific hyper parameter that caused this issue. It appears that when adding back in the subsample = 0.5 line the graph changes slightly to the graph on the left but with the addition of only the scale_pos_weight hyperparameter the graph on the right is seen:





Now it is necessary to address the issue of using zfit. Once again a virtual conda environment was set up for using zfit as the directories clashed resulting in the code timing out when run on the computer. The environment was set up as below:

```
(base) paolominhas@Paolos-MacBook-Air-4 PythonScripts % conda create -n zfit_env -c conda-forge python=3.9 zfit
```

Many packages were downloaded over the 10 minutes it took to lead up. Then once inside the new environment, it was necessary to download the other packages required:

```
(base) paolominhas@Paolos-MacBook-Air-4 PythonScripts % conda activate zfit_env
(zfit_env) paolominhas@Paolos-MacBook-Air-4 PythonScripts % pip install uproot mplhep
Requirement already satisfied: uproot in /Users/paolominhas/opt/anaconda3/envs/zfit_env/lib/python3.9/site-packages (5.6.3)
```

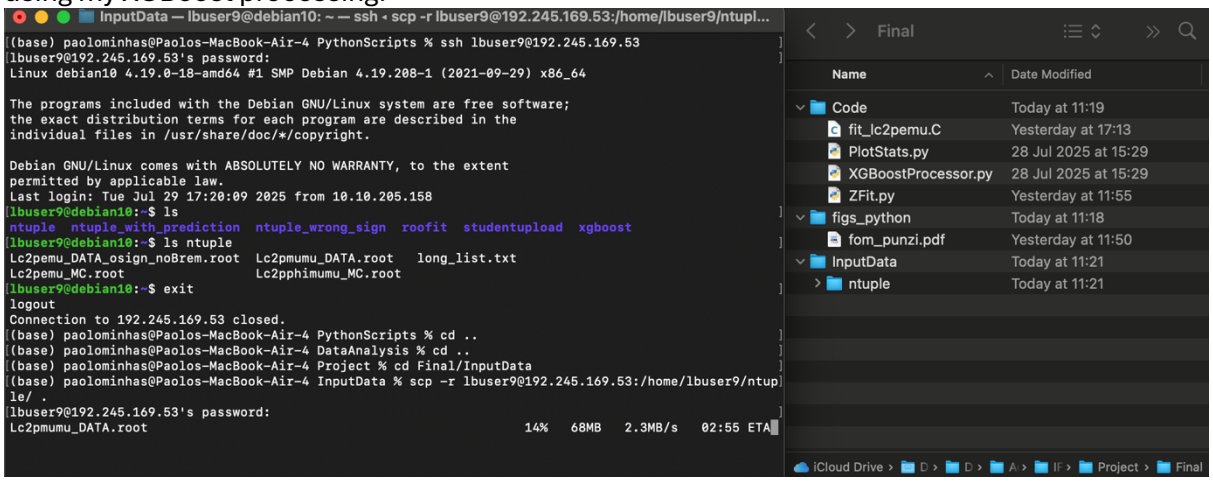
Which actually in this case were all already satisfied. Then the file could be run:

```
(zfit_env) paolominhas@Paolos-MacBook-Air-4 PythonScripts % python3 ZFitting.py
/Users/paolominhas/opt/anaconda3/envs/zfit_env/lib/python3.9/site-packages/zfit/__init__.py:63: UserWarning: TensorFlow warnings are by default suppressed by zfit. In order to show them,
set the environment variable ZFIT_DISABLE_TF_WARNINGS=0. In order to suppress the TensorFlow warnings AND this warning, set ZFIT_DISABLE_TF_WARNINGS=1.
  warnings.warn(
```

Giving the same initial warning as was received when running outside of this environment

Wed 30 Jul

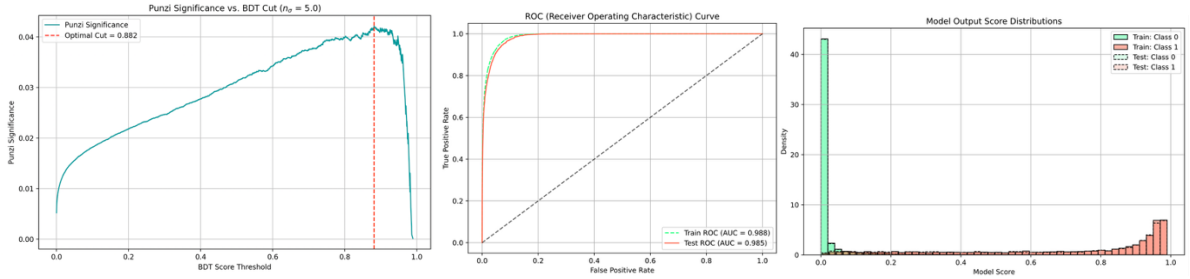
It now becomes important to finalise the data and methods. The C++ file works to provide the final graphs but the python file I have written still has errors regarding the size of arrays. This must be dealt with today. Furthermore, the input files to the final data plots are not from my simulation but rather from Prof. Witek's so this must be altered too. Major changes are needed on the PowerPoint so far as there are no figures and a lot of jargon. Firstly, the input files of the C++ file will be addressed. The first issue is not understanding why 3 input files are used including the reference file. This can be changed however but I will have to first process the reference data using my XGBoost processing.



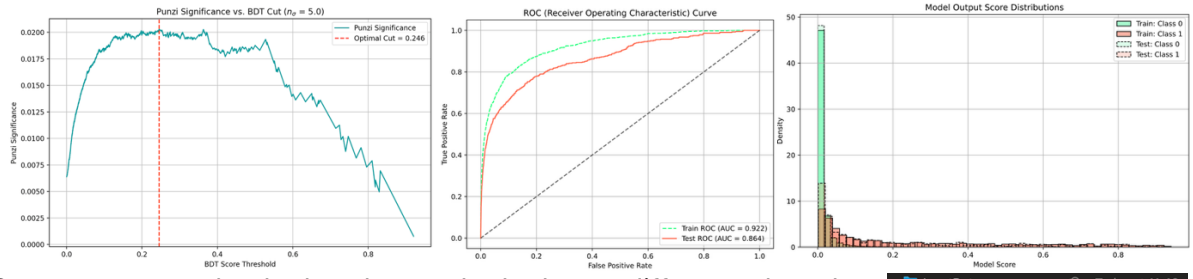
This first step was done promptly – the files were first downloaded from the server (these are the input file for the muon decay reference channel as above. After the download it was necessary to rename the folders and move things around. Now the new data files were passed through the XGBoostProcessor.py file changing the inputs to the mu mu data, and new output names:

```
    plt_stat.pdfplot("XGBoostPlots.pdf", X_train_selected, X_test_selected, y_train, y_test, final_bdt, selec
    plt_stat.SaveBDTResults(final_bdt, selected_feature_names, output_filename="BDTOutputRef.root")
    plt_stat.SaveBDTResultsSmall(final_bdt, selected_feature_names, output_filename="BDTOutputSmallRef.root")


def main():
    print("Starting TMVA-like analysis using XGBoost...")
    #XGB(*Processing("Lc2pemu_MC.root", "Lc2pemu_DATA_osign_noBrem.root", "DecayTree", "long_list.txt"))
    XGB(*Processing("Lc2pmumu_MC.root", "Lc2pphimumu_DATA.root", "DecayTree", "long_list.txt"))
```

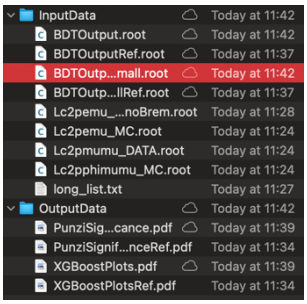
Now this data can be examined. The output plots appeared as follows:



These plots show some excessive overtraining – but will be brought up in today's meeting as the plots with these settings seem to work well for our data set. Just to check the fit data was rerun:



So, we can see clearly that the results look very different when the code is run with the data for our actual decay. Once again, the answer will be determined to this later. Now all the files were moved into input and output folders as required and the second stage of analysis could be run. This involves putting the BDT Output ROOT ntuple files through the fit_lc2pemu.C processing file to get the output plots.



Theory

Machine Learning

ROOT

ML in Python

The Physics of this Interaction

The LHCb Detector

Offline Method Write Up

In this report, we aimed to recreate how the LHCb collaboration searches for a rare decay such as the $\Lambda_c^0 \to p\mu^{\pm}e^{\mp}$ decay seen in this paper. The invariant mass peak is very small compared to other decays from the Λ_c^0 state and the other background. This means several methods are combined to attempt to isolate a signal. The input data from LHCb has already gone through several layers of processing, the first of which is the trigger system. As the full amount of data collected by the detector is too large to be stored offline, a trigger sorts events into well known processes and 'interesting' processes. This means the uninteresting data events are discarded as soon as possible to minimise the disk space that is used (the rate of data intake can be up to 40 MHz. The trigger is divided into three layers: the first of which is hardware and the latter two software. The hardware reduces the event rate of the data to around 1 MHz, before two high-level triggers (software) reduce the rate to 110 kHz and then 12.5 kHz. The high-level trigger software (which is made up of two C++ programs) first selects for events if one or two recognisable signatures (such as dimuon) are visible. Then the second level improves the quality of the track reconstructions via taking effects such as

The first of the methods is the reduction of the large data set through cuts.