Machine Learning Applications in High Energy Physics and Dark Matter Search

A thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

by

Viraj Thakkar

to the

School of Physical Sciences

National Institute of Science Education and Research

Bhubaneswar, India

21/04/2019

DECLARATION

I hereby declare that I am the sole author of this thesis in partial fulfillment of the requirements for a postgraduate degree from National Institute of Science Education and Research (NISER). I authorize NISER to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signature of the Student

Date: 14th May 2019

The thesis work in the thesis entitled Machine Learning Applications in High Energy Physics and Dark Matter Search was carried out under my supervision, in the School of Physical Sciences at NISER, Bhubaneswar, India.

Signature of the thesis supervisor

School: Physical Sciences

Date: 14th May 2019

ACKNOWLEDGEMENTS

I wish to thank Prof. Bedangadas Mohanty for giving me the opportunity to carry on with this exciting new subject of Machine Learning and its applications in High Energy Physics and Dark Matter search. I also wish to thank my guide Sourav Kundu, who has helped me a lot with the analysis part, concepts and in general for this entire project. I wish to thank Vijay Iyer, who was my guide for the SuperCDMS analysis. Thanks a lot for clearing my doubts, making me familiar with the SuperCDMS experiment, and keeping my work organized. I wish to thank Brett Cornell from California Institute of Technology and Robert Calkins from Southern Methodist University for helping me with SuperCDMS analysis and data sources. Finally, I wish to thank all my Experimental High Energy Physics lab members at NISER. Thank you to all for your kind support. I highly appreciate it.

ABSTRACT

The purpose of this project is to use Machine Learning to improve the results in Experimental High Energy Physics (ALICE-LHC) and Dark Matter Search (SuperCDMS) experiments. The first part describes the analysis where we have used supervised Machine Learning algorithm called Boosted Decision Trees (BDT) to improve the significance of resonance particle K^{*0}. The second part describes the analysis on SuperCDMS data where we have done Fiducial Volume optimization and classified Electron Recoils and Nuclear Recoils events by using BDT algorithm for classification.

Contents

List of Figures

List of Tables

Chapter 1

Introduction to Resonances in High Energy Physics

1.1 Motivation

Resonances are short lived(unstable) particles that undergo decay by strong interactions. Life time of these particles are in the order of $\approx 10^{-23}$ seconds. Thus, even if these particles travel at the speed of light, they can only cover distances of the order 10^{-15} meters which is less than the diameter of proton. It is found that the decay products of resonances form a large fraction of final state particles. Resonances contribute to the understanding of hadron production in heavy ion collisions. Due to their short life time(a few fm/c) which covers typical life time of fireball produced in heavy ion collision, a significant fraction of resonances decay inside the hot and dense medium and their hadronic daughters interact with the medium during fireball expansion. That is why, resonances are a sensitive probe of hadronic phase in the dynamical evolution of the fireball and measurement of their yields provides reference for tuning event generator inspired by Quantum Chromodynamics(QCD).

K^{*0} is a short-lived spin one resonance particle having life time $\approx$ 4.2 fm/c. Due to short lifetime of K^{*0}, it can be used as a probe of the hadronic phase created in heavy ion collisions. In non-central heavy ion collisions, in the presence of large initial angular momentum and magnetic field, K^{*0} which is a spin one hadron and can be polarized. Hence, K^{*0} is a useful candidate to understand the effect of initial state angular momentum and magnetic field. All these noble phenomena in heavy ion col-

lision require precision measurement of K^{*0} yield. In the experiment, generally, K^{*0} is reconstructed via invariant mass technique of it's hadronic decay daughters. The invariant mass distribution of hadronic decay daughters of K^{*0} contains K^{*0} signal along with a large combinatorial background mainly coming from the contribution of other primary particles and decays of resonances. The left panel of Figure 1.1 shows the invariant mass distribution of unlike charged (signal plus background) Kπ pairs along with normalized like-sign pairs and mixed event background. Right panel shows K^{*0} signal after background subtraction. From the left panel of Figure 1.1 it is clearly visible that the K^{*0} signal has suffered from a large combinatorial background which gives a low value of significance in K^{*0} signal extraction.

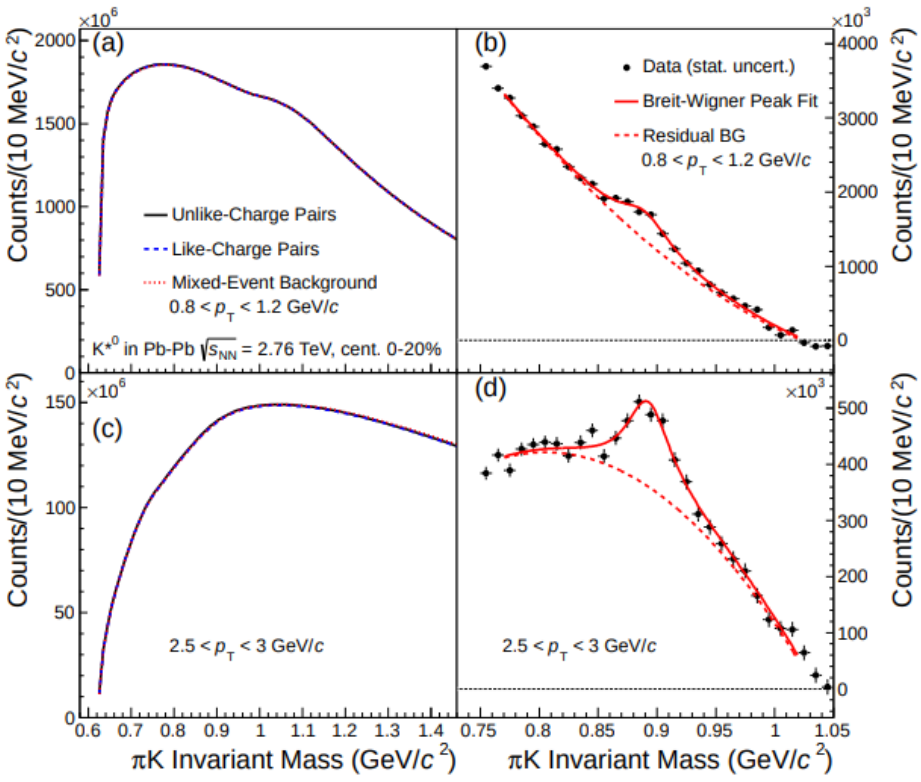

Figure 1.1: K^{*0} signal extraction in Pb-Pb at $\sqrt{s_{NN}}$ =2.76 TeV [1].

1.2 Aim

The aim of this project is to use Machine Learning techniques for the purpose of classification of signal and background events to obtain K^{*0} signal. This is in context of improving the signal to background effects in reconstructing these resonance particles. In the next chapter, we will provide a brief introduction to Machine Learning.

Chapter 2

Machine Learning and TMVA

2.1 Machine Learning

Machine Learning(ML) is a sub-field of Artificial Intelligence(AI) that uses statistical techniques to provide computer systems the ability to "learn" from data, without being explicitly programmed. Machine Learning is used in many real world applications like Optical Character Recognition, Natural Language Processing, Medical diagnosis, for Regression and Classification analysis, etc [2], [3], [4].

2.1.1 Definitions

1. "Field of study that gives computers the ability to learn without being explicitly programmed." -Arthur Samuel.

2. A well-posed Learning Problem -Tom M. Mitchell:
   A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.
   Example:

   (a) **T** : Classifying emails as spam or not spam.

   (b) **E** : Watching you label emails as spam or not spam.

   (c) **P** : The number(or fraction) of emails correctly classified as spam/not spam.

2.1.2 What does it mean to learn?

Learning is a process by which the ML algorithm "looks" at the data and tries to get a "feel" of the trends in the data. The algorithm trains itself by learning the different features present in the data. The algorithm has gained experience over the data after it has finished training over a certain data set, and it has learned about the general trends in the data. It can use this experience i.e. the knowledge about the data it has already learned, and apply it over new data points to make predictions about them.

Illustrative example: Suppose we show a cat and a dog to a toddler and tell him which one is which. We do so for several days by showing the toddler hundreds of cats and dogs. The toddler learns about the various features of the animals like shape of the face, size of the body, shape of the mouth, length of whiskers, the sound it makes, etc. Thus, after gaining enough training, the toddler can classify a new animal and call it a cat or a dog based on its past experience. The toddler has thus "learned" how to classify cats and dogs by training itself over a number of examples.

2.2 Supervised Learning

Supervised learning is the Machine Learning task of making a classifier which maps data to a specific output [5]. In the case of Supervised Learning Classification, we have labels for each data points. Here labels mean to which class or category that particular data point belongs to. We may refer to these labels as $y = 0$ or $y = 1$ for each data point. For instance, referring to our example of the toddler, we can label the cat as $y = 1$ and the dog to can be labeled as $y = 0$. In this project, we will be using Supervised Machine Learning classification.

2.3 Terminologies

2.3.1 Features

Features are the variables which define our data. They are the measure of certain attributes by which the data point can be distinguished from other data points. These features together define the class (i.e., the label) to which the data belongs. We may also refer to them as discriminating variables, as the classifier makes predictions based on the discriminating power of these features.

Let $x_1, x_2, \ldots x_n$ be n number of features, where each feature is a measure of some attribute of the data. We define the feature vector $\mathbf{x}$ to be of the form

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

Here, $\mathbf{x} \in \mathbb{R}^n$ is a n-dimensional feature vector. We assign a label y to the feature vector $\mathbf{x}$. This label y (say $y = 0$ or 1) determines the class to which that data belongs to. Thus, collectively we represent data in the form of its feature vector $\mathbf{x}$ and its corresponding label y as $(\mathbf{x}, y)$.

2.3.2 Classifier

Classifier is the ML algorithm that gives predictions of the labels of $\mathbf{x}$. We denote the classifier output as $h_\theta(\mathbf{x})$. This is nothing put the predicted labels ($y_{predicted}$) given to $\mathbf{x}$ by the classifier $h_\theta(\mathbf{x})$. Here subscript θ denotes the weights which the classifier has learned from the training data set. When $h_\theta(\mathbf{x}) = y(h_\theta(\mathbf{x}) \neq y)$ we say the classifier has correctly(wrongly) predicted the label of $\mathbf{x}$.

2.3.3 Training and Testing data set

Suppose we have m' number of examples i.e. data points. Let us denote the first example by $\mathbf{x}^{(1)}$. That is $\mathbf{x}^{(1)}$ can be represented as follows:

$$\mathbf{x}^{(1)} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ \vdots \\ x_n^{(1)} \end{bmatrix}$$

Thus, the examples $\mathbf{x}^{(1)}$, $\mathbf{x}^{(2)}$ ... $\mathbf{x}^{(m')}$ constitute our data set. We randomly divide our data set into two parts (say in a 60:40 ratio). Let m number of data points be used by the classifier to learn the weights associated with those data points. We denote this sample of m data points as our ***training set***. This training set is the set of data points used by the classifier $h_\theta(\mathbf{x})$ to learn and produce their corresponding weights. The remaining $m' - m$ data points are used to check the performance of the classifier on data points which were not used for training the classifier. This is known as the ***testing set***. As the true labels, y for this set is already known, we can check the performance of the classifier output $h_\theta(\mathbf{x})$ on these testing data points. If the classifier is observed to perform well only on the training set and has low accuracy in classifying events in the testing set, the classifier is said to suffer from the problem of ***Overfitting***. This phenomenon can happen if we have used a lot of features or combination of features for our training purpose.

2.3.4 Application phase

After the classifier has finished with training and testing of data, we are left with the weights which the classifier has learned. The classifier then uses these weights to give predictions $h_\theta(\mathbf{x})$ on data points whose labels are unknown. Thus, with these predictions, we can classify new data points into categories of $y = 1$ or $y = 0$.

2.4 Toolkit For Multivariate Analysis (TMVA)

TMVA provides a framework for multivariate analysis (ML classification algorithms) implemented in ROOT [6]. All multivariate techniques in TMVA belong to the family of supervised learning algorithms. The data set is in the form of signal and background events stored in trees of a root file. TMVA splits the data set for training and testing and thus performs the classification of the events. The output "weights" are stored in an XML file which is then used in the application phase. The classifier then scans through the events in the application phase and predicts them as signal or background using the weights stored in the file [7]. For our analysis, we will be using the *Boosted Decision Trees* (**BDT**) classifier [8].

2.5 Decision Trees

What is a Decision Tree?

A Decision Tree(DT) is a tree-like structure that uses a branching method to illustrate the possible outcome of a decision. An event is either classified as signal or background by either passing or not passing a condition(cut) on a specific node until a decision is made. In order to determine these conditions(cuts), the decision tree is grown starting from the root node.

2.5.1 Building a Decision Tree

The training of a decision tree is the process that defines the splitting criteria for each node. Starting with the root node, an initial splitting criteria for the full training sample is determined. The split results in two subsets of training events and each event go through the same algorithm of determining the next splitting. This procedure is repeated until the whole tree is built. At each node, the split is determined by finding

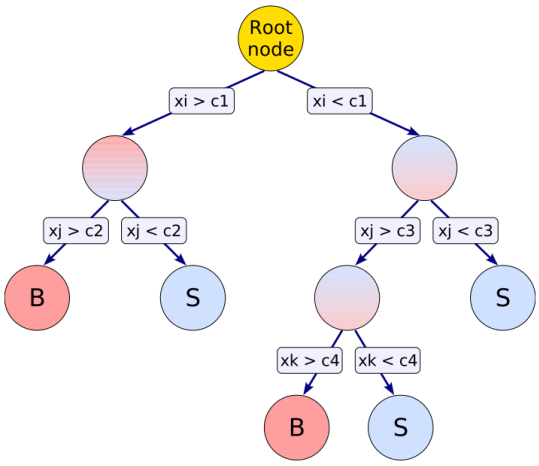

Figure 2.1: A schematic view of a Decision Tree. Beginning from the root node, a sequence of binary splits using the discriminating variables(features) x_i is applied to the data. Each split uses a variable such that at this node we get the best separation between signal and background when being cut on. The leaf nodes at the bottom end of the tree are labeled "S" for signal and "B" for background depending on the majority of events that end up in the respective nodes.

the feature and corresponding cut value that provides the best separation between signal and background. The node splitting stops once it has reached the minimum number of events which is specified in the BDT algorithm (`NEventsMin`).

The leaves, i.e. the final nodes are classified according to their purity $p = S/(S+B)$. If $p > 0.5$ then it is signal and if $p < 0.5$ then it is background. The quality of separation is defined by the so-called impurity function. In our case, we use the *Gini Index*: $p(1-p)$.

The splitting criterion is always a cut on a single variable. The training procedure selects the variable and cut value that optimizes the increase in the separation index between the parent node and the sum of the indices of the two daughter nodes, weighted by their relative fraction of events. The cut values are optimized by scanning over the variable range.

Some cons of decision trees are that if the DT grows too big in size, it can lead to

overfitting. DTs are sensitive to overfitting. Usually, an ensemble of decision trees is required for better performance.

2.6 Boosting

Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules. Boosted Decision Trees (BDT) is a prediction model which uses an ensemble of "weak learners" (typically decision trees). It is a model designed to make fewer and fewer mistakes as more trees are added to it. By weak learners, we mean the classifier whose training accuracy is just better than 50%. We will be using the *Adaptive Boost* (**adaboost**) algorithm for this project [9].

Model Description: There is an ensemble of small decision trees built by training. Each tree attempts to correct errors from the previous tree. Thus, BDT is an algorithm which combines forests of DTs, each weighted according to their importance. We start with the original event weights for the first tree. At each step, we ensure that the weights are normalized. The subsequent tree is trained using a modified event sample where the weights of previously misclassified events are multiplied by a common boost weight w. The boost weight is derived from the misclassification rate, err, of the previous tree and is defined as

$$w = \frac{1 - err}{err},$$

where err is the ratio of total weight of mistakes to total weights of all points.

Let us denote individual classifier of a DT as $h(\mathbf{x})$. Let $h(\mathbf{x}) = 1$ for signal and $h(\mathbf{x}) = -1$ for background. The boosted event classification $y_{Boost}(\mathbf{x})$ is then given by:

$$y_{Boost}(\mathbf{x}) = \frac{1}{N_{collection}} \sum_{i=1}^{N_{collection}} \ln(w_i).h(\mathbf{x})$$

The sum is over all the classifiers in the collection. Small(large) values for $y_{Boost}(\mathbf{x})$ indicate a background-like(signal-like) event. We thus obtain a BDT response value (ranging -1 to 1) for each $\mathbf{x}$.

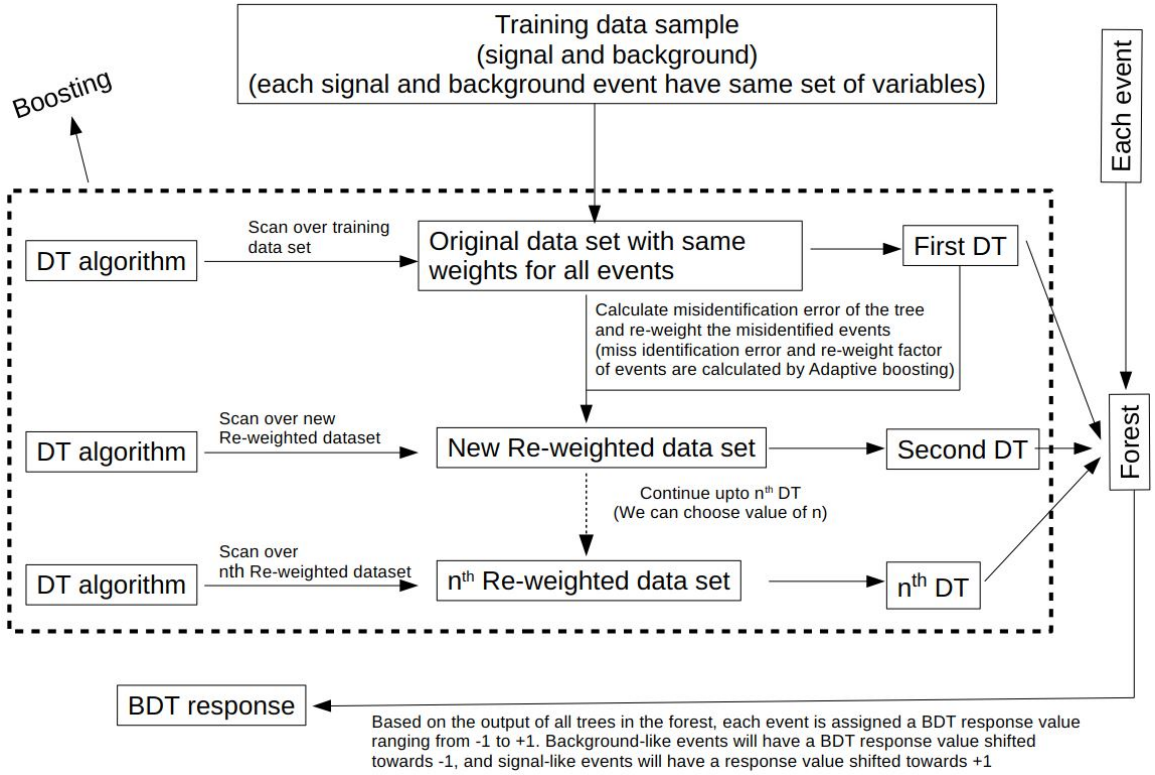

Figure 2.2: This figure shows the flow chart of the BDT algorithm.

Chapter 3

Analysis (ALICE-LHC)

3.1 Data set and selection

3.1.1 Input data and event selection

We have analyzed Monte Carlo(MC) production data from LHC10f6a period to demonstrate signal extraction K^{*0} with Machine learning approach. This MC production is simulated with PYTHIA event generator [10] for $\sqrt{s_{NN}} = 7$ TeV. The generated MC production data is passed through a GEANT3 based simulation of ALICE detector and then reconstructed with ALICE specific reconstruction algorithm. We have used this reconstructed MC data as a proxy of real data for our analysis. We have taken $|V_z| < 10$ cm (z-component of vertex position) for good event selection. After the good event selection cut, the total number of events analyzed are ∼5 million.

3.1.2 Track selection cuts

We have used some standard track selection cuts as used in the ALICE experiment to select the primary track.

Standard ITS-TPC 2010 track cuts are used for this analysis. Details about these track selection cuts are given below:

1. $p_T > 0.15$ GeV/c
2. -0.8 $< \eta <$ 0.8
3. Reject kink daughters.
4. Ratio of crossed rows over findable cluster > 0.8
5. Minimum (Maximum) number of rows crossed in TPC is 70 (159).

6. TPC χ^2/clusters < 4.0
7. ITS χ^2/clusters < 36.0
8. $(DCA)_r$ $(p_T) < 0.0182 + 0.0350 p_T^{-1.0}$ cm
9. $|DCA|_z < 2$ cm
10. $|y_{pair}| < 0.5$

The AliRoot and AliPhysics versions used for this analysis are AliRoot version: v5-09-20-1 and AliPhysics version: vAN-20171201-1. More details on the cuts applied can be found in reference [11].

3.1.3 Particle selection

K^{*0} is reconstructed by invariant mass technique on its hadronic decay daughters. The daughter particle of K^{*0} are kaons and pions, and they can be identified by using Time Projection Chamber (TPC) information [12]. In TPC, particles are identified by the difference between the measured energy loss and the value expected for different mass hypotheses. Both pions and kaons are selected by a cut of $|\sigma_{TPC}| < 2.0$ where σ_{TPC} is the TPC dE/dx (energy loss) resolution. Figure 3.1 shows σ vs. momentum distribution for the selected pion candidates.

3.2 Analysis Description

We have reconstructed the K^{*0} signal by pairing unlike charge K and π in two different ways:

1) **Traditional invariant mass technique**: In this method, we formed an invariant mass distribution of unlike charged K and π pairs from the same event. This unlike charged K and π distribution contains K^{*0} signal along with large combinatorial background. Invariant mass distribution for like sign K and π pair from the same event is used to reproduce the combinatorial background. After subtracting the like

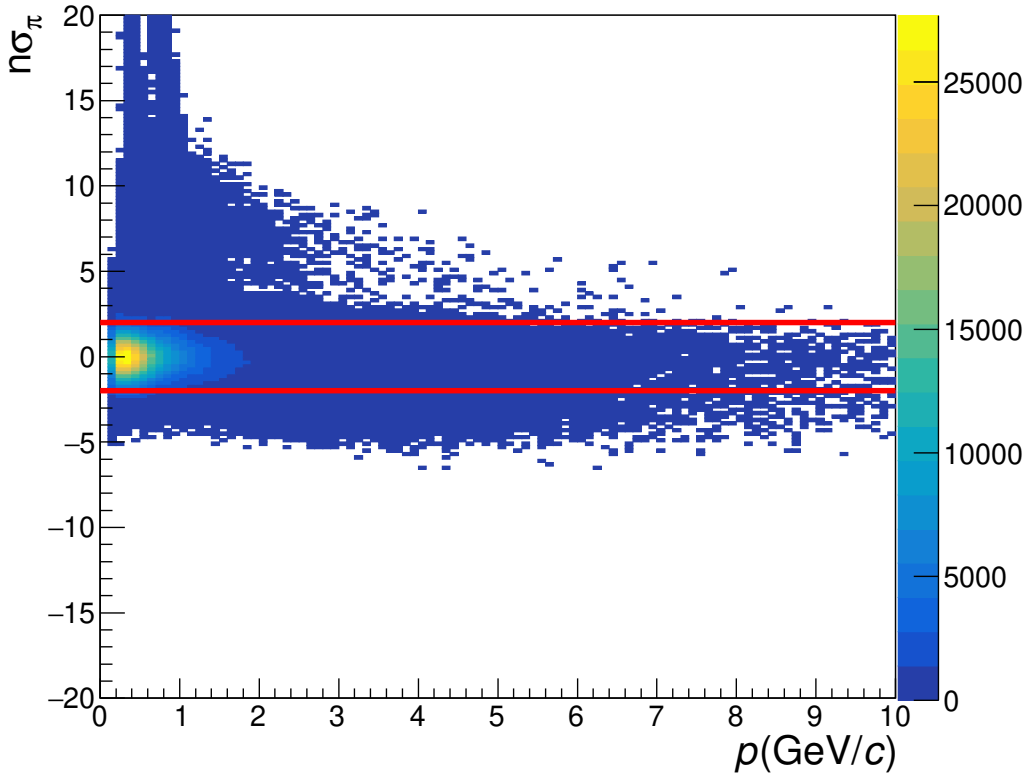

Figure 3.1: σ vs. momentum distribution for selected pion from dE/dx distribution of TPC. Red line corresponds to the applied $|\sigma_{TPC}| < 2$ cut.

sign combinatorial background from invariant mass distribution of unlike charged Kπ pairs, we get K^{*0} signal.

2) **Invariant mass technique along with BDT algorithm**: In this, method before using the traditional invariant mass technique we used BDT algorithm to reduce the combinatorial background from unlike charged Kπ pairs. The detailed analysis description is as follows:

The classification is done with the help of Toolkit for Multivariate Analysis(TMVA) in ROOT. BDT algorithm was used to train the classifier with signal and background events.

For the training purpose, two types of trees are involved: the signal tree and the

background tree. The signal tree consists of all the features describing the signal events (like p_t , η , DCA etc.). The background tree consists of exactly the same features passed through the same preselection cuts as the signal. Classifier cut efficiency curves are obtained which gives the value of BDT response (call it "BDT cut value") for which maximum significance is obtained.

After training and testing, TMVA produces an output weight file which is used during the application phase. For the purpose of application, we construct another tree consisting of all signal and background events. Again these events are made by passing through the same cuts and consists of the same features as present in the training phase. The BDT response value obtained from the training phase is used to separate signal and background events. Events with BDT response greater(less) than the cut value are passed as signal(background) by the classifier.

To obtain the final signal via BDT, we first pass all signal plus background events in the application phase. The invariant mass histogram is constructed for the events which pass the BDT cut. Then we pass all the background events in the application phase. Again the background events with BDT response greater than cut value are filled in an invariant mass histogram(meaning the background which the classifier fails to reject after the BDT cut). Finally, these two histograms are subtracted to obtain the final signal.

3.2.1 Preparation of data for training, testing and application phase

Signal: Signal candidates are formed by true K^+ and π^- pairs which are decayed from K^{*0}. Generated level MC PID is used to select true K^+ and π^-.

Background: Background candidates are formed by the same sign π and K pairs, where these π and K pairs are selected by using $|\sigma_{TPC}| < 2.0$.

Unlike pairs: For the application phase, all possible unlike pair combinations of $K^+\pi^-$ are taken which contains both signal plus background. Unlike sign pairs are constructed by selecting K and π pairs with $|\sigma_{TPC}| < 2.0$.

3.3 Results and Plots

3.3.1 Input features and Training phase

The analysis is done for $p_T < 0.8$ GeV/c of reconstructed mother particle K^{*0}. Let us denote K as the Daughter1 particle and π as the Daughter2 particle. The following features were considered as initial candidates for training BDT:

1. p_T (Transverse momentum) of Daughter 1.
2. p_T of Daughter 2.
3. p_T of Mother.
4. η (Pseudo-rapidity) of Daughter 1.
5. η of Daughter 2.
6. η of Mother.
7. cos θ*(production plane): The angle between the daughter pion and normal of the production plane in the rest frame of the mother where production plane is the plane spanned by the beam axis (Z direction) and momentum of mother track in the laboratory frame.

8. cos θ^*(beam axis): The angle between the daughter pion and beam axis in the rest frame of the mother.

9. DCA$_{xy}$ of Daughter 1: Distance of closest approach between primary vertex and daughter kaon track in xy plane.

10. DCA$_{xy}$ of Daughter 2: Distance of closest approach between primary vertex and daughter pion track in xy plane.

11. Invariant Mass ($M_{K\pi}$).

Figure 3.2 shows distributions of the features for signal and background.

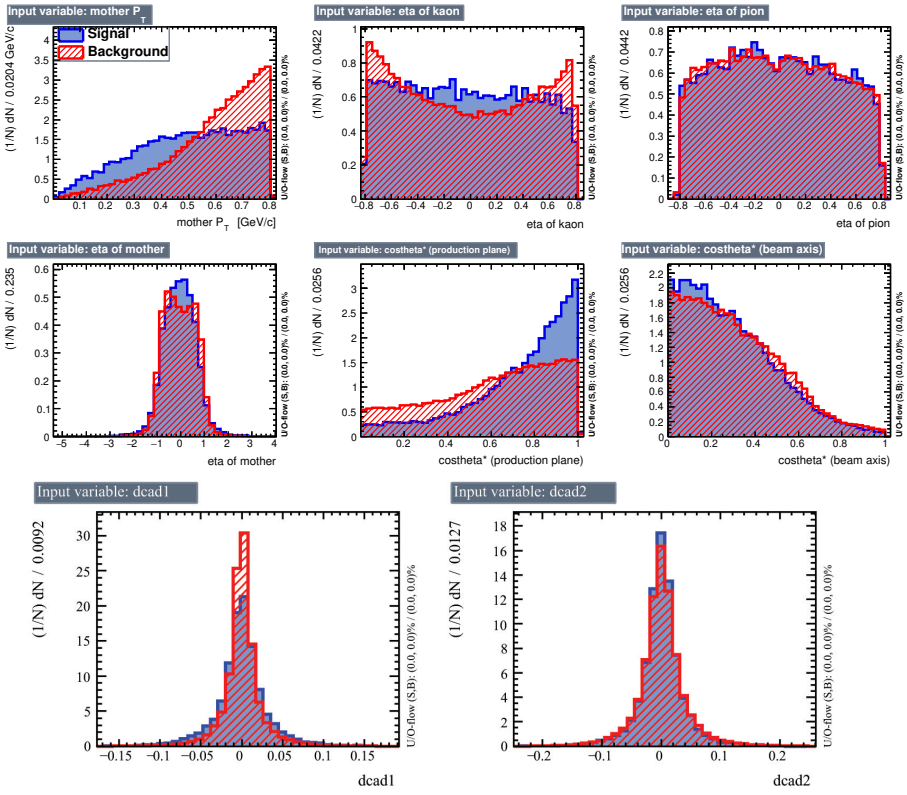

Figure 3.2: Features: Distribution for signal(blue) and background(red). Signal candidates are formed by true K^+ and π^- pairs which are decayed from K^{*0}. Background candidates are formed by same sign π and K pairs.

Figure 3.3 shows the correlation coefficients percentage between all the features. Some features have high correlation with the invariant mass. We therefore remove those features from the training phase to avoid any further bias that could affect the training results.

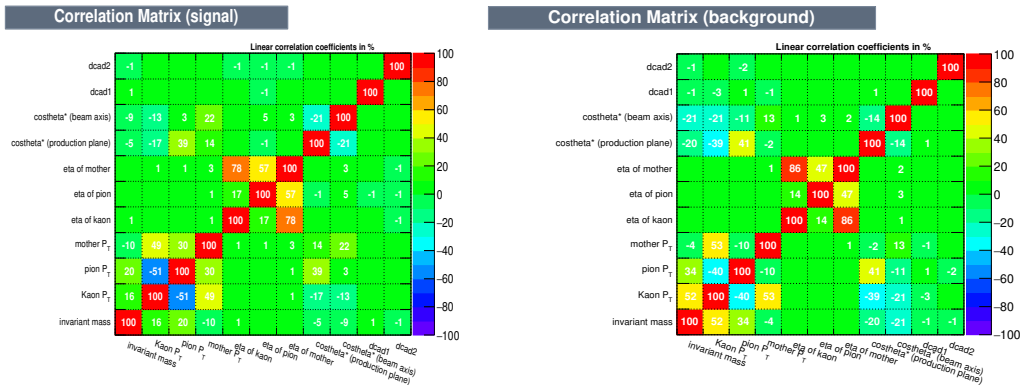

Figure 3.3: Correlation coefficients percentage between all the features for signal and background.

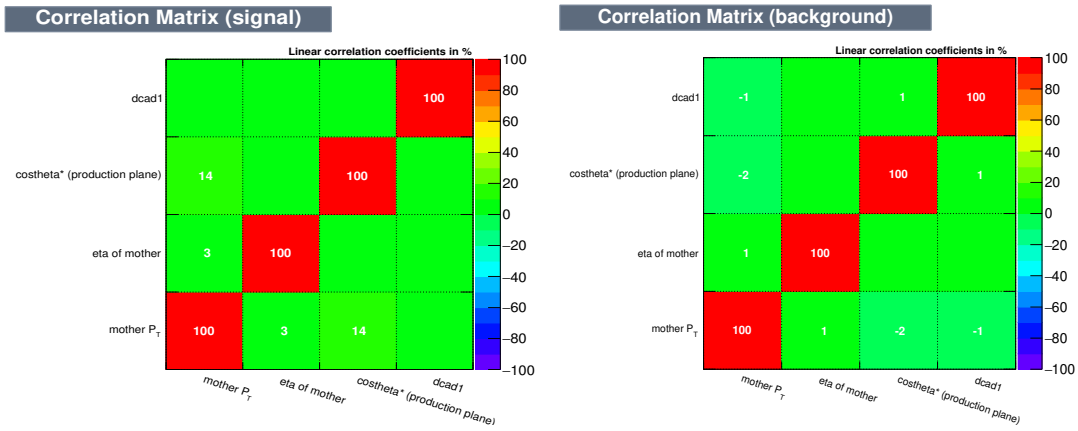

Figure 3.4: Correlation coefficients percentage between selected features for signal and background.

Now that all the features are approximately uncorrelated to invariant mass, we select the top 4 with the highest discriminating power. Refer to Figure 3.4. Thus, we use these features for classification of signal and background. We do not select invariant mass for the training and testing phase.

3.3.2 TMVA training and testing phase output

Training is done using 15000 signal and 50000 background events and the remaining events are used for testing. The following BDT configuration was used:

`NTrees=850:MinNodeSize=2.5:MaxDepth=3:BoostType=AdaBoost:AdaBoostBeta=0.5:UseBaggedBoost:BaggedSampleFraction=0.5:SeparationType=GiniIndex:nCuts=20` [7]

Figure 3.5 shows the BDT response for both the training and testing samples. A low value of Kolmogorov-Smirnov(KS) test for background suggests that the classification algorithm is not efficiently able to separate signal and background events. Nevertheless, a cut value of -0.0794 on BDT response is able to reduce 30% of background with maximum significance and Signal Efficiency of 92% as shown in Figure 3.6.

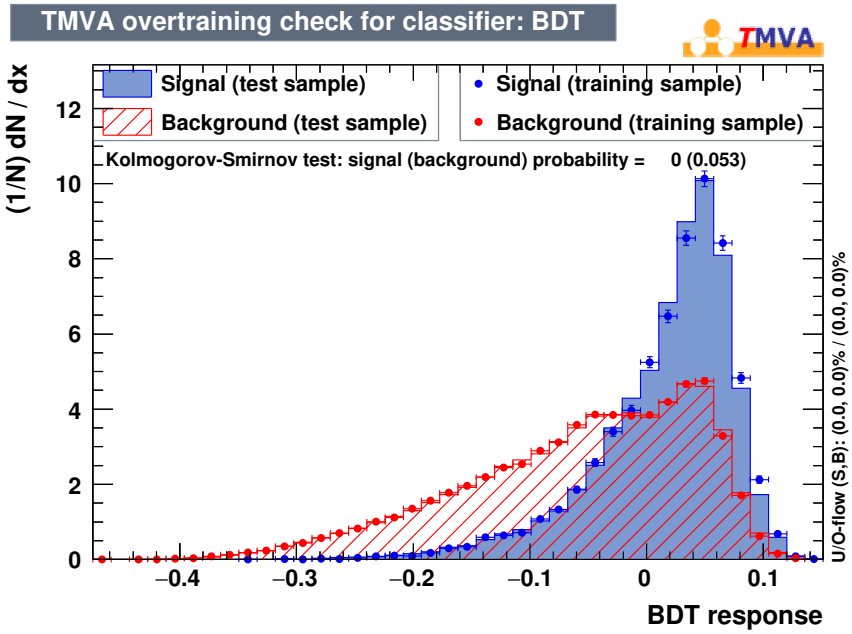

Figure 3.5: Classifier output distribution for signal and background shown for both testing and training events of MC data.

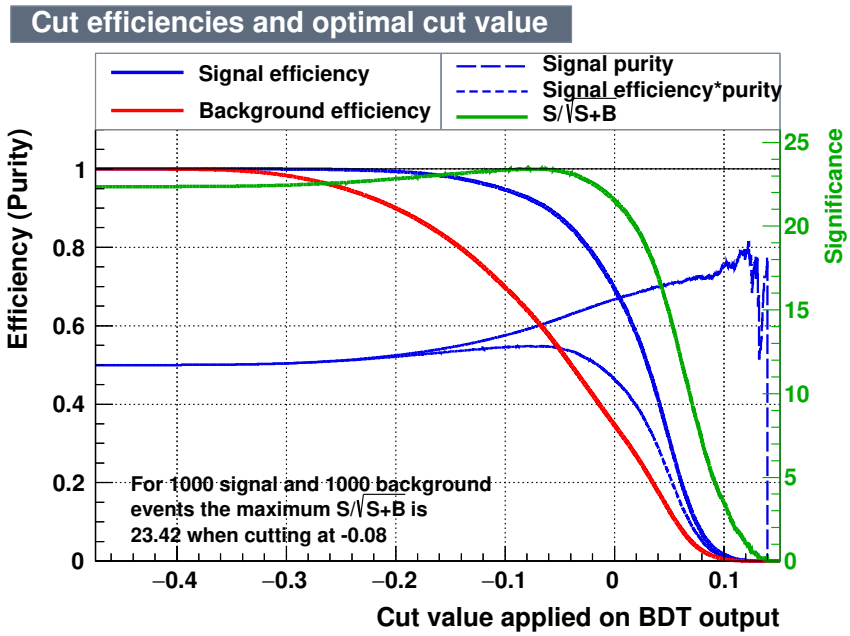

Figure 3.6: Classifier cut efficiency plot as a function of BDT response value. The green curve shows significance. We can see that maximum significance is obtained at BDT cut value ≈ -0.08. We will use this cut value in the application phase.

3.4 Application phase and final signal

We have obtained the BDT cut value(≈ -0.08) from the training phase. In the first step, we apply this BDT cut value on unlike pair events which constitutes the signal plus background candidates. The events which pass the cut value (i.e. for events with BDT response > BDT cut value) are filled in a histogram of invariant mass. Similarly, in the second step, we apply the BDT cut value on the like-sign pairs which constitutes the background candidates. Background events with BDT response> BDT cut value are filled in another invariant mass histogram. Figure 3.7 (left) shows these invariant mass distributions after the BDT cut. Figure 3.7 (right) shows the invariant mass distribution of unlike pairs and like-sign background without using BDT algorithm i.e. the traditional invariant mass technique. To obtain the signal, we subtract the two invariant mass distributions (unlike pairs minus like-sign background) given in

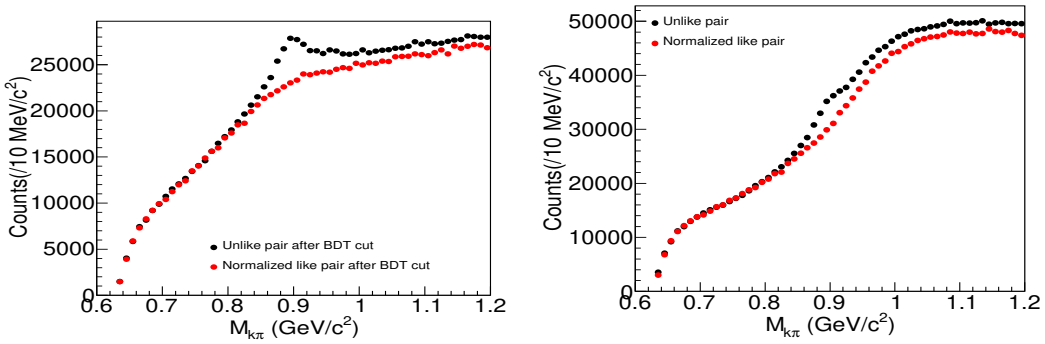

Figure 3.7: Left panel: Normalized invariant mass distribution of like sign pairs along with invariant mass distribution of unlike charge kπ pairs of the like sign and unlike sign candidates which passed the BDT cut value. Right panel: Same as left panel but without applying BDT cut on unlike and like charge pairs.

Figure 3.7 for left and right panel respectively. Figure 3.8 shows like sign background subtracted invariant mass distribution of unlike charged Kπ pairs. We fit the signal with Breit-Wigner + Polynomial 2-degree function, where Breit-Wigner describes the resonance signal and polynomial 2-degree function describes the residual background.

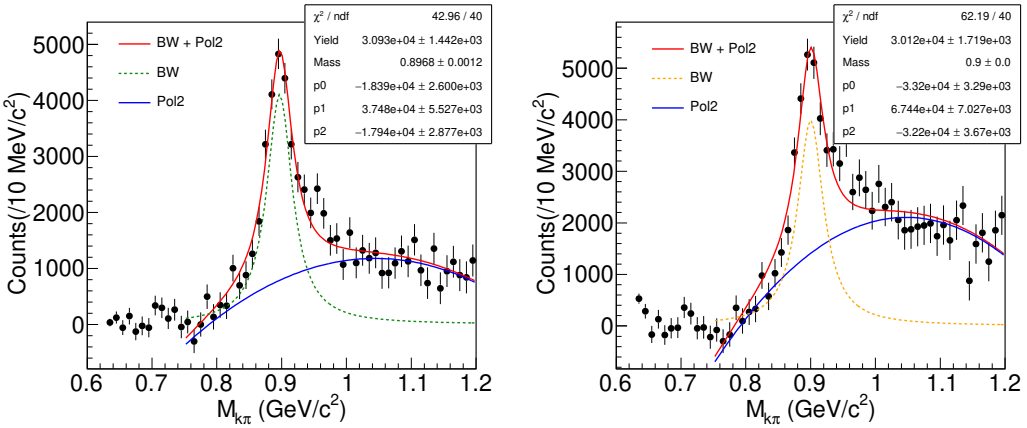

Figure 3.8: Left panel: Like sign background subtracted invariant mass distribution of unlike charged Kπ pairs after applying BDT cut on both unlike charged and like charged pairs, along with the Breit-Wigner +Pol2 fit. Right panel: Same as left panel but without applying BDT algorithm on unlike charge and like charge pairs.

Table 3.1: Comparison between ML BDT method and like-sign background subtraction method. S: Signal, B: Background.

Method	χ^2/ndf	S	S+B	$S/\sqrt{S+B}$	S/B	Yield
BDT	42.96/40	30931.6/0.92=33621.3	906953	32.48	0.0353	34253
Like-sign	62.19/40	30115	1355500	25.87	0.023	34253

Table 3.1 shows the extracted K^{*0} yield for both of these methods along with the significance, signal-to-background ratio and χ^2/ndf of the Breit-Wigner+ Polynomial2 fit of invariant mass distribution of like-sign subtracted unlike sign pairs. K^{*o} yield is corrected in BDT by signal efficiency (=0.92). Extracted yield, significance and χ^2/ndf value of the fit suggests a better extraction of K^{*0} signal is achieved by using Machine Learning approach via BDT algorithm.

3.5 Application on ALICE data

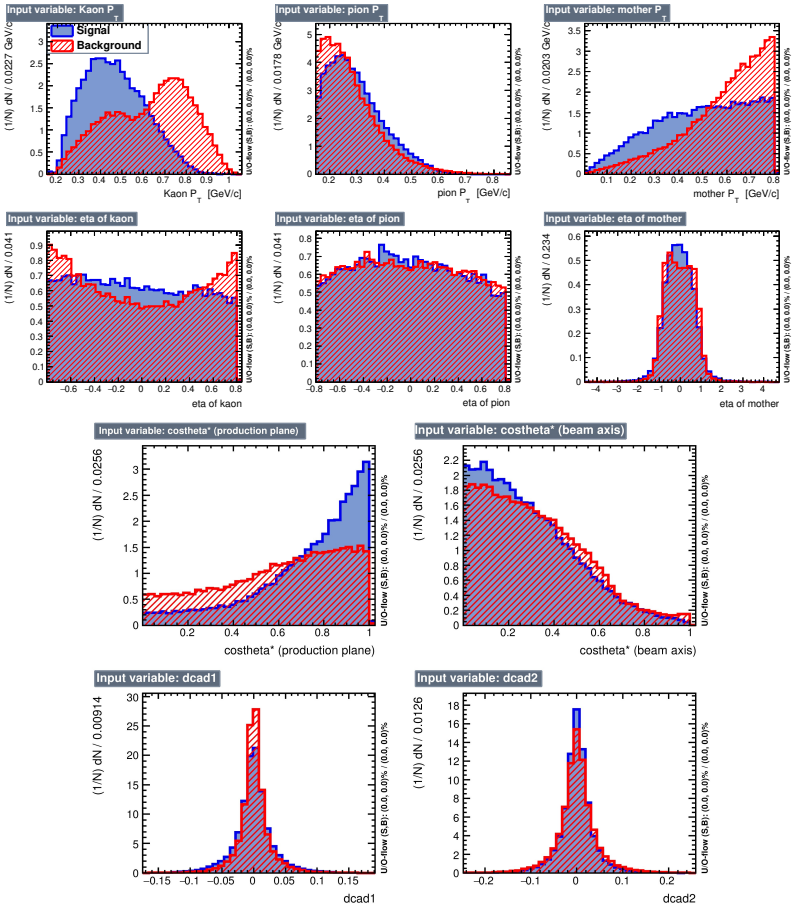

Figure 3.9: Features: Distribution for signal(blue) and background(red). Signal candidates are formed by true K^+ and π^- pairs which are decayed from K^{*0}. Background candidates are formed by same sign π and K pairs.

A similar analysis was done on ALICE data from period LHC10d for pp collision at $\sqrt{s} = 7$ TeV for 10 Million events. The same primary track selection cuts as given in section 3.3.1 were used.

For the training purpose, signal candidates are constructed by true K^+ and π^- pairs which are decayed from K^{*0}. Generated level MC(PYTHIA) PID is used to select true K^+ and π^- events.

Background candidates are formed by same sign π and K pairs, where these π and K pairs are selected by using $|\sigma_{TPC}| < 2.0$ cut from the data.

Unlike sign pairs are constructed by selecting all combinations of K and π pairs with $|\sigma_{TPC}| < 2.0$ from the data.

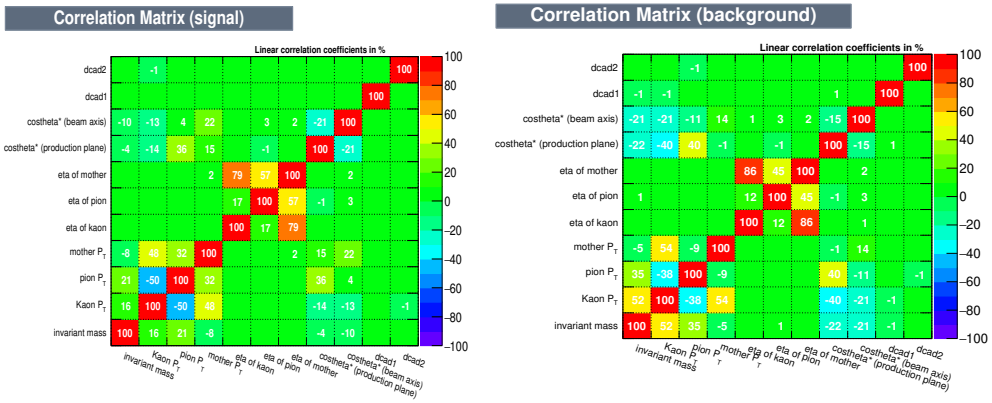

Figure 3.10: Correlation coefficients percentage between all the features for signal and background.

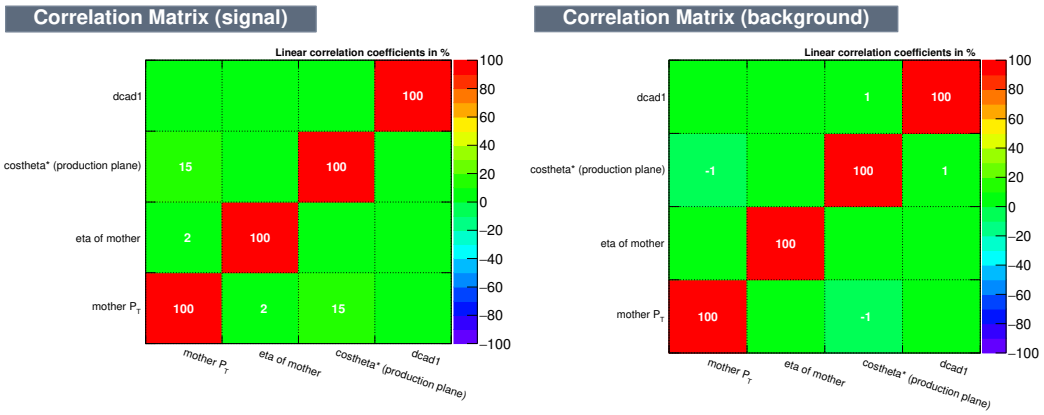

Figure 3.11: Correlation coefficients percentage between features for signal and background.

Figure 3.10 shows the correlation coefficients percentage between all the features. We remove the features with high correlation to invariant mass from the training phase to avoid any further bias that could affect the training results.

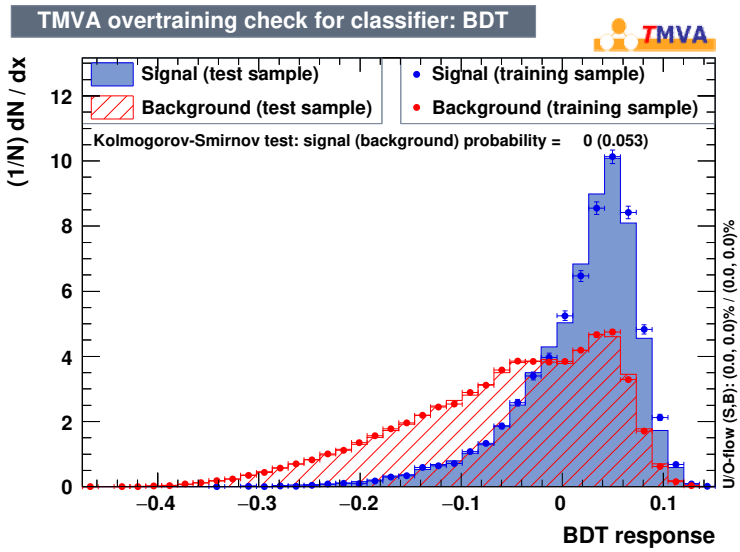

Figure 3.12: Classifier output distribution for signal and background shown for both testing and training events of ALICE data.

Figure 3.11 shows the correlation between the features selected for training. Figure 3.12 shows the BDT response for both the training and testing samples.

In Figure 3.13, we obtain maximum significance at BDT cut value = −0.09. We use this cut value in the application phase. Figure 3.14 (left panel) shows the normalized invariant mass distribution of like sign pairs along with invariant mass distribution of unlike charged kπ pairs of the like sign and unlike sign candidates from data whose BDT response > BDT cut value. Figure 3.14 (right panel) is the same as left panel but without applying the BDT cut on unlike and like sign charge pairs. Figure 3.15 shows like-sign background subtracted invariant mass distribution of unlike charge Kπ pairs, where the left panel is the signal obtained from BDT and the right panel is signal obtained from like-sign technique.

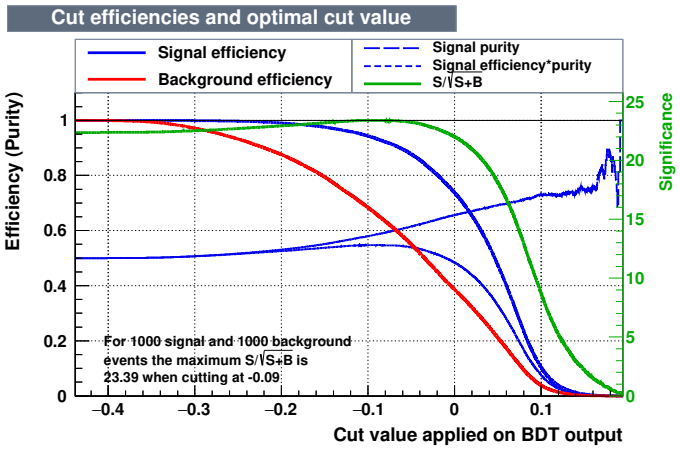

Figure 3.13: Classifier cut efficiency plot as a function of BDT response value. The green curve shows significance. We can see that maximum significance is obtained at BDT cut value ≈ -0.09. We will use this cut value in the application phase.

Table 3.2: Comparison between ML BDT method and like-sign background subtraction method by application to ALICE data. S: Signal, B: Background

Method	χ^2/ndf	S	S+B	$S/\sqrt{S+B}$	S/B
BDT	50.35/46	75934.8/0.93=81650.3	3626890	39.87	0.0214
Like-sign	47.11/43	82346.8	5227420	36.01	0.016

Table 3.2 shows the extracted K^{*0} yield for both of the methods along with the significance, signal-to-background ratio and χ^2/ndf of the Breit-Wigner+ Polynomial2 fit of invariant mass distribution of like-sign subtracted unlike sign pairs. K^{*0} yield is corrected in BDT by signal efficiency (=0.93). Extracted yield, significance, and χ^2/ndf value of the fit suggests a better extraction of K^{*0} signal is achieved by using Machine Learning approach via BDT algorithm for ALICE data.

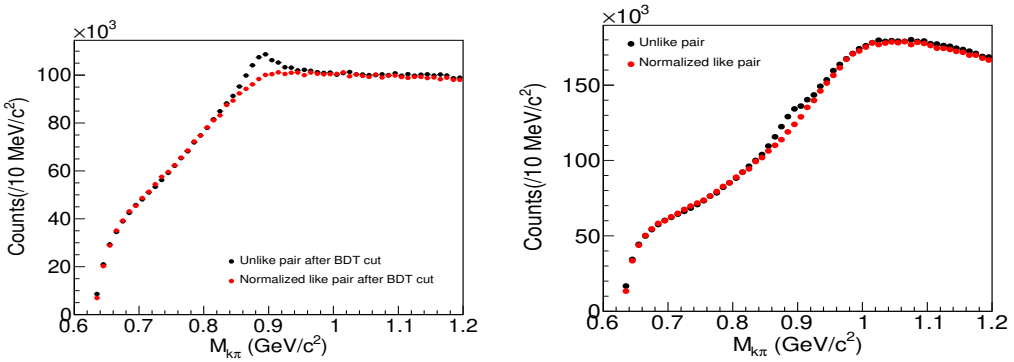

Figure 3.14: Left panel: Normalized invariant mass distribution of like sign pairs along with invariant mass distribution of unlike charged kπ pairs of the like sign and unlike sign candidates which passed the BDT cut value. Right panel: Same as left panel but without applying the BDT cut on unlike and like charge pairs.

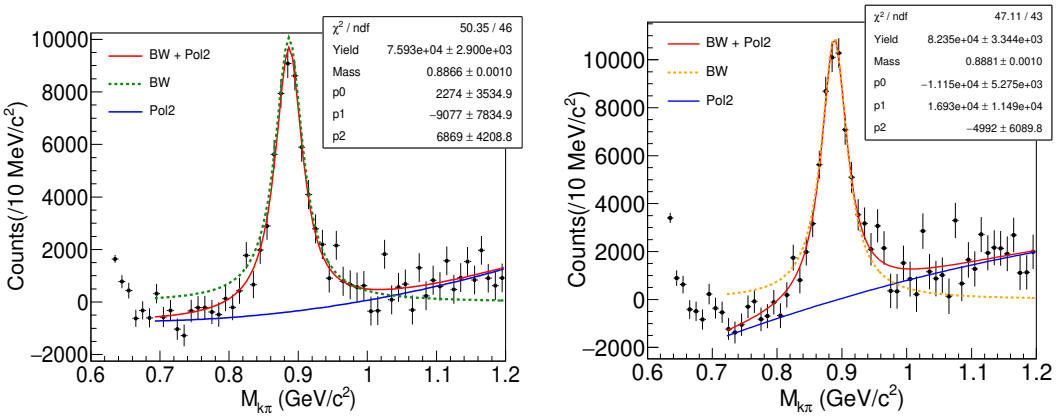

Figure 3.15: Left panel: Like sign background subtracted invariant mass distribution of unlike charge Kπ pairs after applying the BDT cut on both unlike charge and like charge pairs from data, along with the Breit-Wigner +Pol2 fit. Right panel: Same as left panel but without applying the BDT cut on unlike charge and like charge pairs.

Chapter 4

Summary and Conclusions (ALICE)

In this analysis, we have extracted K^{*0} signal from two methods: 1)Traditional invariant mass technique and 2)Invariant mass technique combined with machine learning classification by BDT algorithm. It is found that a better signal extraction is obtained via a combination of BDT and invariant mass technique as compared to only traditional invariant mass method for both Monte-Carlo generated data as well as real data from ALICE detector. Better value of significance is obtained by applying machine learning via BDT(32.48) as compared to the traditional invariant mass technique(25.87) for MC data. Similarly, for ALICE data, we obtain a better significance(39.87) by machine learning technique as compared to like-sign technique(36.01).

4.1 Outlook

In the future, we can extend this analysis for resonances which undergo 3-body decay. Machine learning can also be applied in real data such as data from Pb-Pb collisions where the resonance signal contains a large combinatorial background. This method will be a useful tool in search of rare resonances such as K*(1410), K*(1680) and Ξ(1820).

Chapter 5

Introduction to Dark Matter

Scientific evidence establishes that the universe is composed of mysterious entities like Dark Matter and Dark Energy along with baryonic matter [13], [14]. Astonishingly, calculations from the lambda-CDM model of the universe reveal that the baryonic matter which makes up the Earth, Sun, Stars, and galaxies comprises only 5 % of the mass-energy density of the universe. The remaining part is composed of Dark Energy which accounts for 69 % and Dark Matter which accounts for 26 % [15]. Dark Matter is a hypothetically proposed form of invisible matter that makes up about 85 % of the total matter in the universe. The nature of Dark Matter and the way it interacts is still unknown and thus remains one of the major unsolved problems in physics. Dark Matter interactions are rare because they do not interact with electromagnetic radiation and hence are difficult to detect by direct observations. Detecting this mysterious form of matter and studying its properties can open new frontiers in physics.

5.1 Evidence for Dark Matter in the Universe

Several scientific evidence has led us to hypothesize the existence of a non-luminous type of matter to account for the 'missing mass' in the universe. We will briefly summarize some of these observations as follows:

5.1.1 Velocity Dispersions

The Virial theorem states that for a closed bounded system, the time-averaged kinetic energy follows the relation:

$$\langle K.E \rangle = \frac{1}{2} \sum \langle F.r \rangle \,, \tag{5.1}$$

where F is the Force. For a potential that depends only on the distance r between the particles in the form of $V(r) \propto r^n$, we find the following relation to hold

$$2 \langle K.E \rangle = n \langle V \rangle \,, \tag{5.2}$$

where $\langle V \rangle$ is average the potential energy of the system and $\langle K.E \rangle$ is the average kinetic energy of the particles. We have $n = -1$ in the case of Gravitational potential. These results work in a regular system like our solar system and thus, are expected to work on the larger scales of galaxies. We can consider a galaxy to be composed of stars of mass m_p and orbiting an enclosed mass M_{tot}. It can be easily shown that the gravitational potential follows a relationship of the form

$$\langle V \rangle \approx \alpha \frac{G M_{tot} m_p}{R} \tag{5.3}$$

The kinetic energy is of the form

$$\langle K.E \rangle = \frac{1}{2} m_p v^2 \tag{5.4}$$

where α is a constant depending on the mass distribution of the system and v represents the velocity. Using the Virial theorem, we get

$$v^2 = \alpha \frac{G M_{tot}}{R} \tag{5.5}$$

Hence, using Eq. 5.5, we can find measurements of velocity distributions as a function of distance from the orbit center and thus, calculate the mass enclosed by

that orbit. For the purpose of astronomical measurements, we can write $v = 2\pi R/T$, where R is the radius and T is the time period. Hence, for a typical planet and Sun system, we have

$$\frac{4\pi^2 R_{planet}^2}{T_{planet}^2} = \frac{GM_{sun}}{R_{planet}} \tag{5.6}$$

Hence, we get the Kepler's Third law of planetary motion in Newtonian gravity as

$$R_{planet}^3 = \frac{GM_{sun}}{4\pi^2} T_{planet}^2 \tag{5.7}$$

There have been observations that velocity dispersion estimates of elliptical galaxies do not match the predicted velocity dispersion from the observed mass distribution [16].

5.1.2 Galaxy Rotation Curves

The arms of a spiral galaxy rotate around its galactic center. We expect the orbital velocity to fall off as $1/r$ for stars near the edge of the galaxy, and thus the enclosed mass will reach an asymptotic value according to Eq. 5.5. On the contrary, the galaxy rotation curve remains flat as the distance from the center increases [17].

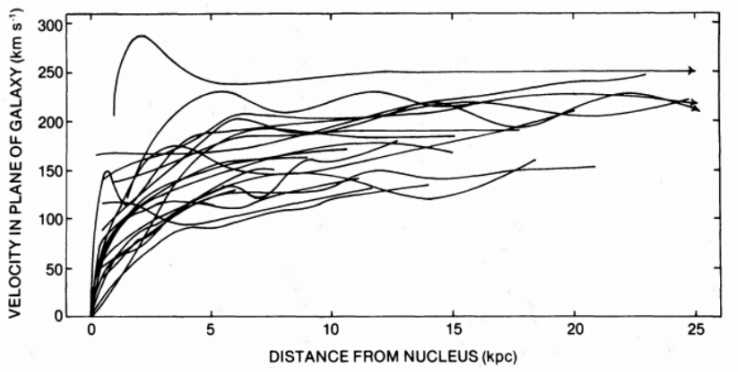

Figure 5.1: Plot of the rotation curves measured from Rubin *et. al.* [17]. It shows that all curves reach their asymptotic flat value.

5.1.3 Gravitational Lensing

As a consequence of Einstein's general relativity theory, massive objects like galaxy clusters can act as a lens and bend the light coming from a distant source to an observer. The extent of lensing increases with the mass of the massive object. Hence, by measuring the distortion parameters, the mass of the intervening cluster can be calculated. The mass-to-light ratio obtained by these calculations can only be explained by the existence of non-luminous dark matter densities in the cluster [18]. It is important to note that dark matter itself does not interact with the electromagnetic radiation i.e. light. Dark matter causes the space-time to bend and thus results in the lensing effect.

5.1.4 Cosmic Microwave Background Radiation

Dark Matter affects the Cosmic Microwave Background Radiation(CMB) through its gravitational potential. The most compelling evidence for particle dark matter is achieved by application of Einstein's field equation 5.8 from General Relativity in the cosmological model called ΛCDM, where Λ stands for dark energy which is approximately 75 % of the energy density of the universe in the present epoch, and CDM stands for Cold Dark Matter.

$$R_{\mu\nu} - \frac{1}{2} g_{\mu\nu} R = -\frac{8\pi G_N}{c^4} T_{\mu\nu} + \Lambda g_{\mu\nu} \tag{5.8}$$

Here $R_{\mu\nu}$ and R are Ricci tensor and scalar respectively, $g_{\mu\nu}$ is the metric, G_N is the Newtonian gravity constant, $T_{\mu\nu}$ is the stress-energy tensor, and Λ is the cosmological constant. Further derivations and information on the ΛCDM can be found in reference [16]. Measurements of the cosmic microwave background (CMB) capture the energy densities and allow us to use CDM cosmology to project primordial densities forward in time and thus, measure the current mass density in the universe [19].

5.2 Detection of Dark Matter

Dark Matter is hypothesized to be composed of non-baryonic subatomic particles. Millions of such particle pass through every square centimeter of Earth per second. Dark Matter particles are known to interact on the scale of the weak force with baryonic matter. Thus, many experiments aim to test this hypothesis.

5.2.1 Weakly Interacting Massive Particles

Weakly interacting massive particles (WIMPs) are hypothetical particles that are thought to constitute dark matter [20]. WIMP is a new elementary particle which interacts via gravity and other non-Standard Model forces on a scale similar to that of the weak force.

5.2.2 The SuperCDMS experiment

The SuperCDMS collaboration aims to detect WIMPs(Weakly interacting massive particles) which are possible candidates for dark matter. The underground detector in the Soudan Mine can detect the rare signal of WIMP when it collides with the atomic nucleus of the detector. The usefulness of such an experiment strongly depends on the detector's ability to distinguish between the WIMP signal and other sources of background particles coming from the cosmos.

5.2.3 SuperCDMS detector

The experiment aims to measure recoil energy from the elastic scattering of Dark matter particles with the target nucleus in the crystal. The detectors measure the ionization and phonons produced by the interaction of particles in their crystal. These measurements determine the energy deposited in the crystal in each interaction. It also gives information about the kind of particle in the event depending on the energy

deposited. The ratio of ionization signal to phonon signal differs for particle interactions with atomic electrons ("electron recoils (ER)") and atomic nuclei ("nuclear recoils(NR)"). The vast majority of background particle interactions are electron recoils. WIMPs and neutrons are expected to produce nuclear recoils. This allows WIMP-like scattering events to be identified even though they are rare compared to the vast majority of unwanted background interactions. More information on the detectors and related terminologies can be found in the references [21], [22] and [23].

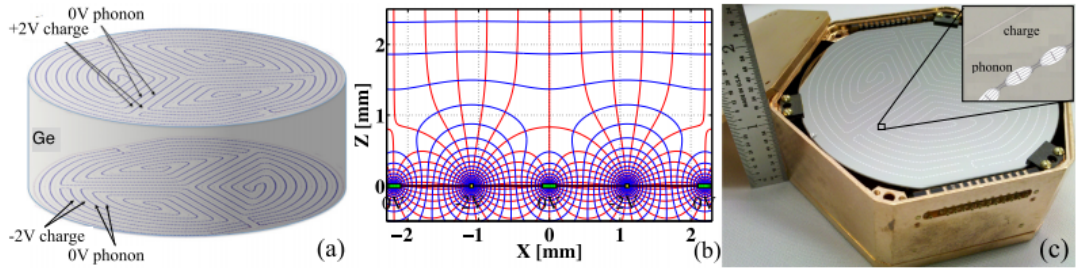

FIG. 1. (a) Phonon and ionization sensor layout for iZIP detectors deployed at Soudan. The Ge crystal is 76 mm in diameter and 25 mm thick. Both faces are instrumented with ionization lines (one face with +2 V and the other with −2 V) that are interleaved with phonon sensors (0 V) on a ∼1 mm pitch. The phonon sensors are arranged to give 4 phonon readout channels for each face, an outer sensor surrounding three inner ones. (b) Magnified cross section view of electric field lines (red) and equipotential contours (blue) near the bottom face of a SuperCDMS iZIP detector. The −2 V ionization electrode lines (yellow) are narrower than the 0 V athermal phonon collection sensors (green). (c) Fabricated iZIP detector in its housing.

Figure 5.2: SuperCDMS detector. Figure taken from [23].

5.3 Objective of Analysis

The objective of this analysis is to obtain a better separation between Nuclear recoils(NR) and Electron Recoils(ER) by using Supervised Machine Learning algorithms. Better separation of NR events from the rest of the background can provide information on the events which can be possible candidates for dark matter.

Chapter 6

Analysis (SuperCDMS)

6.1 Signal and Background Datasets

6.1.1 Preselection Cuts

- Not Empty: Removes empty events in the detector with no trigger.
- Good Flash time: Removes events when the detector was not flashed for period greater than 3300 seconds.
- Base Temperature: Removes events or short period of events collected when the base temperature was not in the good range.
- Voltage Bias: Ensures voltage was maintained with 4V range.
- Good Start time: Removes events with Non Stationary Optimal Filter delay in seconds from the global trigger.
- Analysis threshold cut of 10 keV on phonon recoil energy.

6.1.2 Data source

The data was collected during the period 07/10/2013-07/17/2014, which is called Run 134. Data was taken from the Stanford underground facility from the path `/data/R134/dataReleases/Prodv5-3-5/merged/all/`. Events taken were from Cf calibration data for signal and Ba data for background.

6.1.3 Signal

Our signal data constitutes of Nuclear recoils (NR) events. We have used ^{252}Cf calibration data as the primary source of NR events. WIMPs and neutrons are expected to produce Nuclear recoils. Since dark matter candidates WIMPs have not been confirmed yet, we rely on the theoretical WIMP model and use the standard Event rate spectrum as calculated by Lewin and Smith [24]. This WIMP spectrum for Dark Matter mass of 100 GeV/c^2 is shown in Figure 6.1 by the blue curve. This spectrum is corrected by the detector efficiency, which is shown in the same plot(red) with a different y-axis. The corrected spectrum(black) represents the spectrum that we may observe if we were to have a Dark Matter source near our detector. The green colored spectrum represents the Cf calibration data. We have to correct the differences in distributions for the Cf spectrum(green) and the WIMP-corrected spectrum(black) by constructing weight vector distributions so that the Cf distribution will mimic the signal. The weight vector distribution is calculated by taking the ratio of the distributions of WIMP-corrected spectrum and the Cf spectrum, as shown in Figure 6.2. The Cf spectrum has to be corrected using these weights to represent the signal from WIMPs. This constitutes our WIMP model [25].

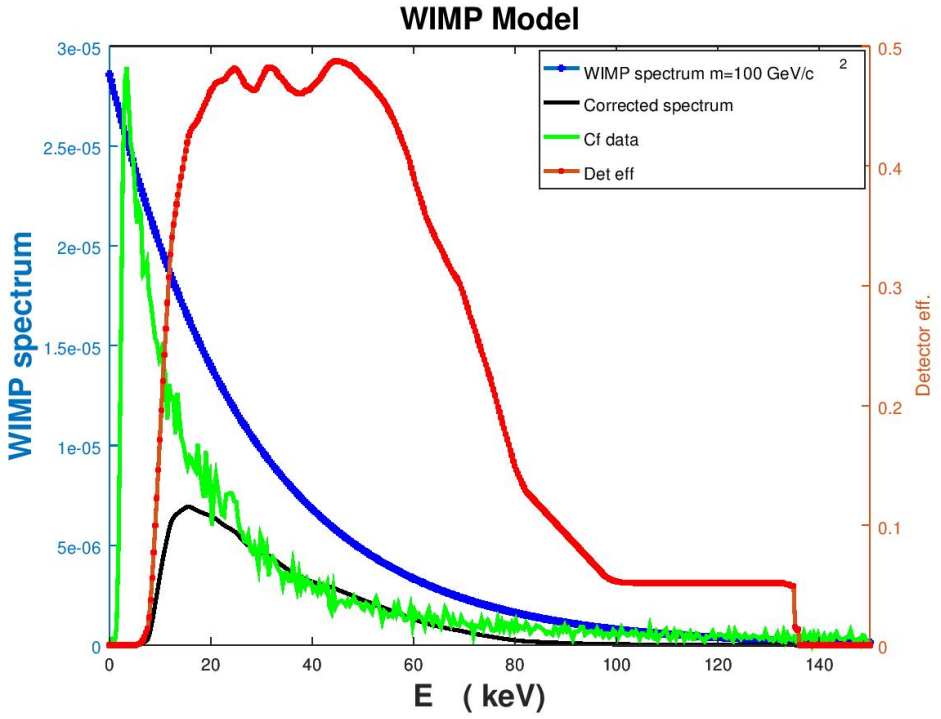

Figure 6.1: WIMP signal model.

6.1.4 Background

We have used electron recoil events from the ^{133} Ba calibration data to represent our background events. Ba is a source of gamma rays, which indeed produces electron recoils in the detector. Also, we have taken the mean average inner charge to be positive for our analysis to remove zero charge events i.e. (qi1OF + qi2OF)/2 > 0 (See Appendix for definition).

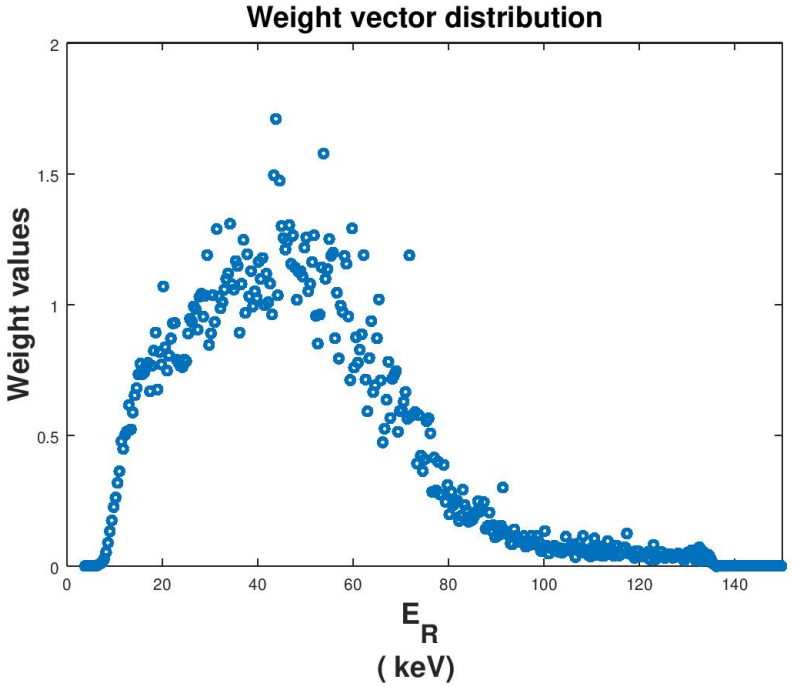

Figure 6.2: Distribution of weight vector.

6.2 Fiducial Volume Optimization

Fiducialization is a process which is implemented to remove unwanted surface events which are collected by the detector. These unwanted surface events lead to misidentification of Electron Recoils with suppressed ionization collection as Nuclear Recoils near the surface of the detector. Hence, we have to remove the radially outer and surface events as they may not have proper charge collection. Radially outer events may have a different electric field due to the fringe effect of the copper detector housing. Surface events may not get full amplification. A combination of the event parameters defines a "Fiducial Volume" (FV) of the detectors. Fiducial Volume Optimization can be thought as a process to remove the vast majority of the physical region of the detector, which produces yield suppressed electron recoils that mimic nuclear recoil signal. Events inside the fiducial volume are called "bulk" events and events outside

the fiducial volume are called "surface" events. Now we will set the fiducial volume across three different parameter space as described below.

6.2.1 Phonon Radial Cut

We set the Phonon Radial cut in the parameter space of phonon radial partition using outer phonon channel (prpartOF) and the total phonon energy (ptNF). Projections to Y-axis (prpartOF) for each ptNF bin is taken which gives the prpartOF distribution. Figure 6.3 which shows the distribution for prpartOF in the range of 1.0 keV $<$ ptNF $<$ 1.1 keV. We obtain the mean(μ) and standard deviation(σ) by fitting Figure 6.3 with a Gaussian distribution. By repeating the same process for each ptNF bin, we obtain the $\mu + 2\sigma$ data points which are shown in the Figure 6.4 in the prpartOF vs ptNF phase space. We refer to this band as the "2-σ band". Finally, this 2-σ band is fitted with an exponential function which sets the Phonon radial cut. According to the definition (See Appendix), prpartOF is the ratio of outer phonon sum and total phonon sum energy. This implies that radially outward events will have higher values of prpartOF. Hence, events lying above the exponential curve are rejected by the Phonon Radial cut.

6.2.2 Charge Symmetric Cut

Charge Symmetric FV cut is defined in the parameter space of inner charges collected from Side 1 and Side 2 of the detector, i.e. qi1OF and qi2OF as shown Figure 6.5. The interleaved electrode geometry of the detector was designed to have an asymmetric charge collection for events near the surface and symmetric charge collection for events in the bulk region of the detector. Based on this understanding, we select events in which the charges collected by the two sides of the detector (S1 and S2) do not differ significantly from each other, hence the name "symmetric cut". Thus, we optimize our

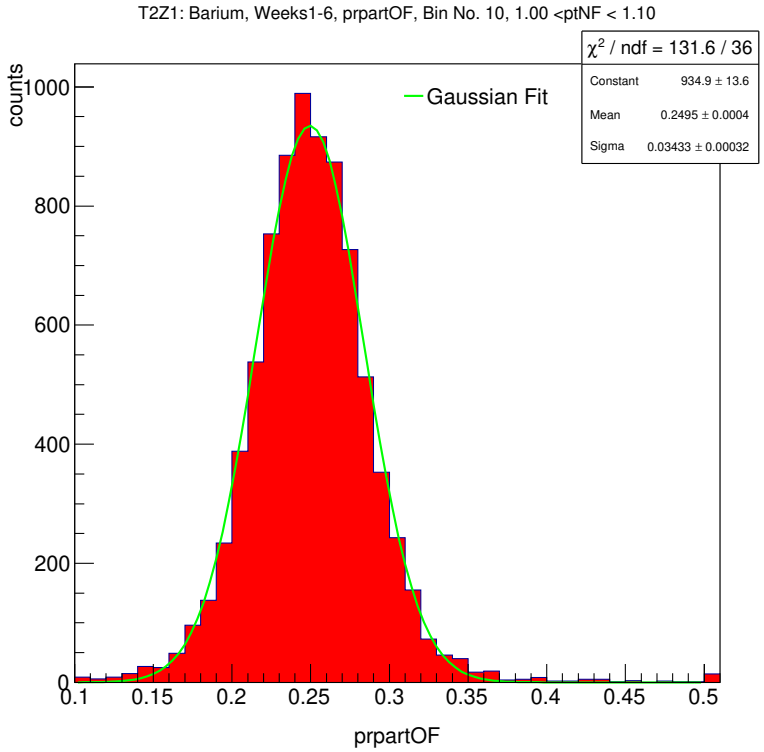

Figure 6.3: Distribution of phonon radial partition (prpartOF) obtained by taking projection for a ptNF bin in the range 1.0 keV $<$ ptNF $<$ 1.1 keV. This distribution is fitted with a Gaussian function(green) to obtain the mean and standard deviation.

cut by creating $\mu \pm \sigma$ bands for the events lying around the symmetric $y = x$ line in the qi2OF vs. qi1OF parameter space. This is done by taking projection along the Y-axis and fitting the distribution of Inner charge(S1) with a Gaussian function (Similar steps as in Phonon Radial cut). Additional, we introduce a horizontal threshold cut on qi2OF and a vertical threshold cut on qi1OF to complete our definition region of Charge Symmetric cut. Events outside this region denote asymmetry in charge collection on the two sides of the detector and hence, such events are rejected.

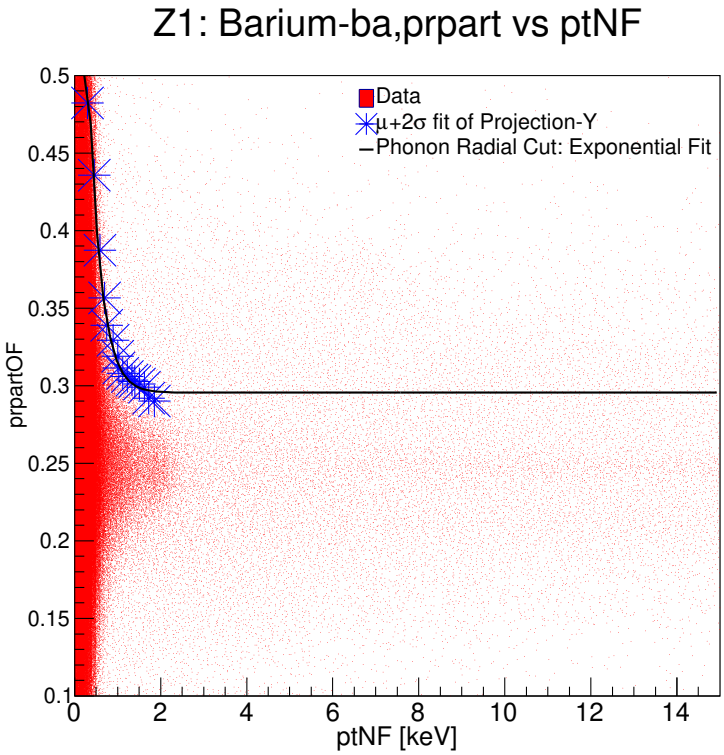

Figure 6.4: Distribution of Barium data in the prpartOF vs ptNF phase space (Red). The blue markers represent the "2-σ" band obtained from the Gaussian fitting of the prpartOF distribution for each ptNF bin. An exponential function of the form $e^{-a_0+a_1 x} + a_2$ (where a_0, a_1 and a_2 are the fitting constants) is used to fit the data points of the "2-σ" band. This exponential fit (black) sets the definition of our Phonon Radial Cut. Events above the phonon radial cut are rejected.

6.2.3 Charge Radial Cut

Regions closer to the cylindrical wall (also referred to as the sidewall) of the crystal do not result in full charge collection, and hence, we have to remove the events closer to the outer edge of the detector. Thus, events with higher values of outer charge as compared to inner charge are rejected by this cut. The Charge Symmetric cut is defined in the Outer Charge vs. Inner Charge parameter space for both Side 1 and Side 2 of the detector. Steps similar to the ones described above are followed to obtain the "3-σ" band which passes over the charge noise blob as seen in Figure 6.6.

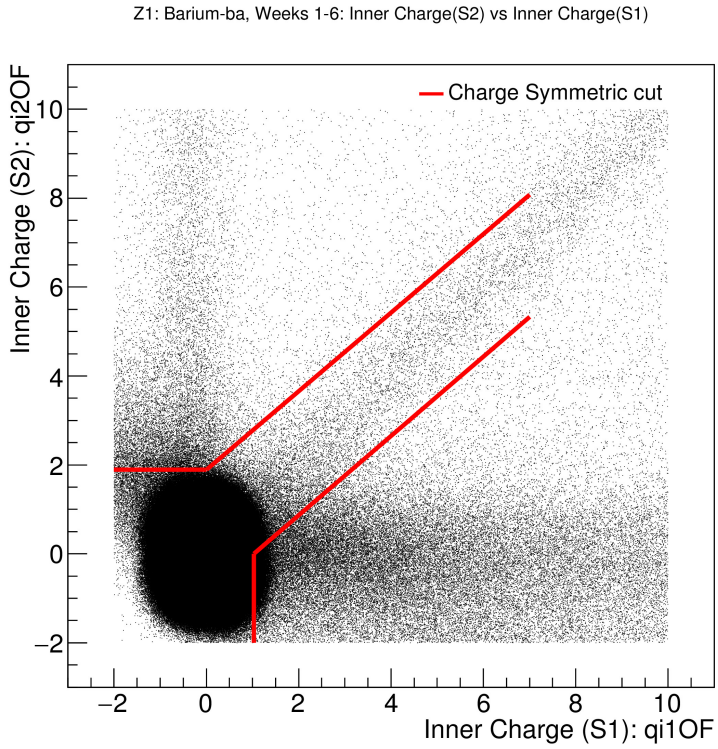

Figure 6.5: Distribution of Barium data in the qi2OF vs qi1OF phase space (Black). The Charge Symmetric Cut is drawn in red.

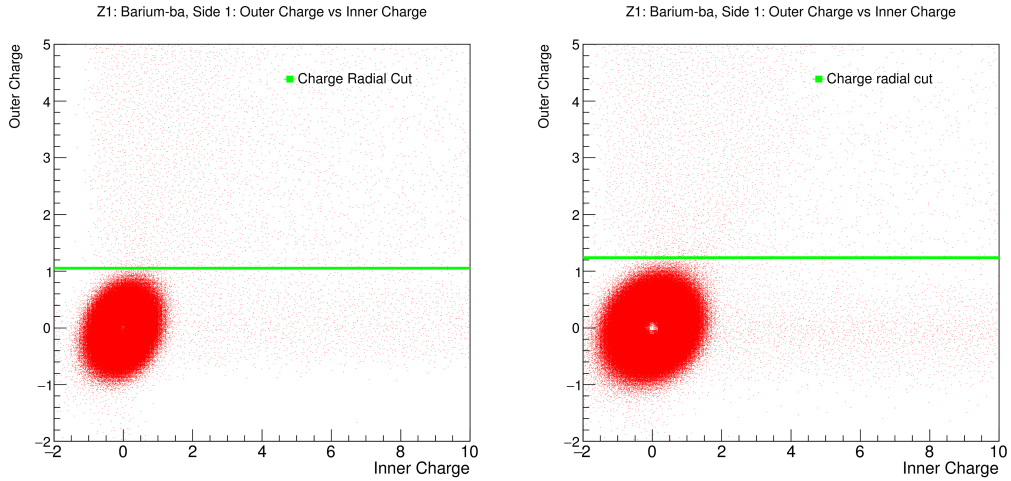

Figure 6.6: Distribution of events for Barium data in the Outer charge vs. Inner charge parameter space for both sides 1 and 2 of the detector. The Charge Radial Cut is shown in green. Events above the green line are rejected.

Chapter 7

ML Application and Results (SuperCDMS)

For training our ML BDT algorithm, we have used Cf data as a source of NR events and Ba data as a source of ER events.

7.0.1 Input features for ML training phase

The following input features were used for training our BDT as shown in Figure 7.1:

- precoiltNF: the non-luke phonon energy (ptNF-plukeqOF).
- ytNF: qsummaxOF/precoiltNF
- qzpartOF: Primary face fiducialization parameter, charge z partition.
- qrpart1OF: Primary sidewall fiducialization parameter.
- pzpartOF: Bulk z position proxy.
- prpartOF: Bulk r position proxy.
- qsummaxOF: qsummaxOF is maximum qsum.

Figure 7.2 shows the classifier cut efficiencies for our BDT training. We select the value of BDT cut $= -0.0528$ to maximize our separation of ER and NR events in the application phase of ML.

Figure 7.3 shows the BDT response for both the training and testing samples. A good separation is obtained between ER and NR events because of taking mutually exclusive cuts in the definition of the signal and background training sample.

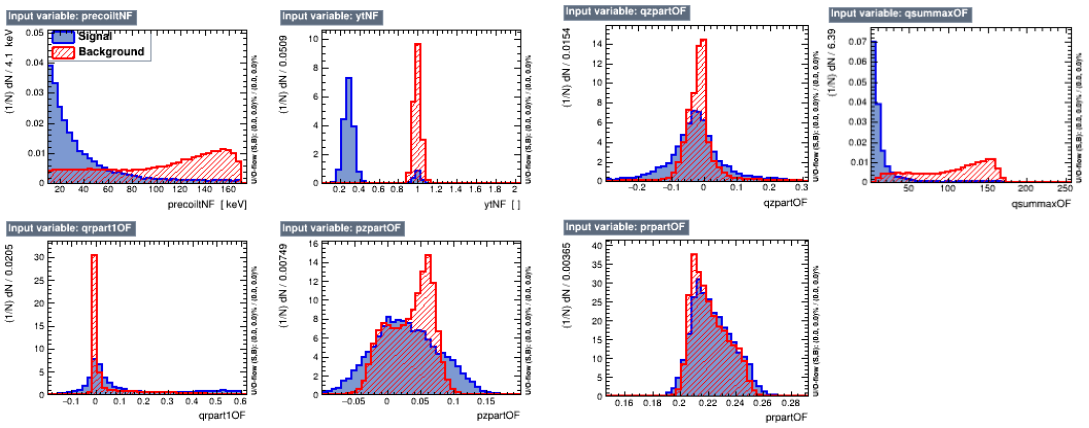

Figure 7.1: Distribution of input features used for training of BDT.

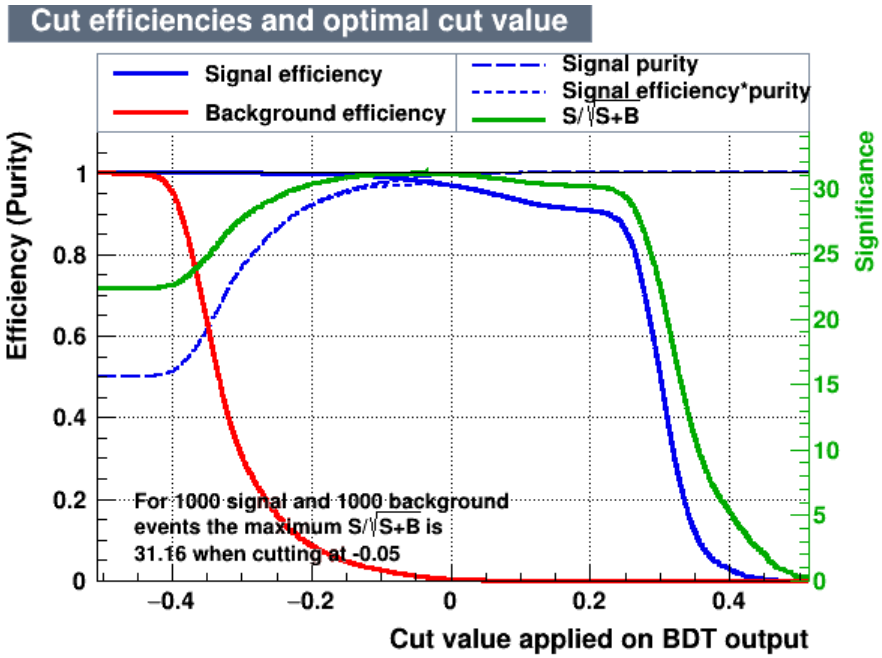

Figure 7.2: Classifier cut efficiency plot as a function of BDT response value. The green curve shows significance. We can see that maximum significance is obtained at BDT cut value ≈-0.0528. We will use this cut value in the application phase.

7.0.2 Application phase

We have taken Cf data, which consists of both ER and NR events in the application phase of ML. By setting a BDT cut of -0.0528, events with BDT response value greater than the BDT cut value are classified as signal(NR in this case) and those failing to

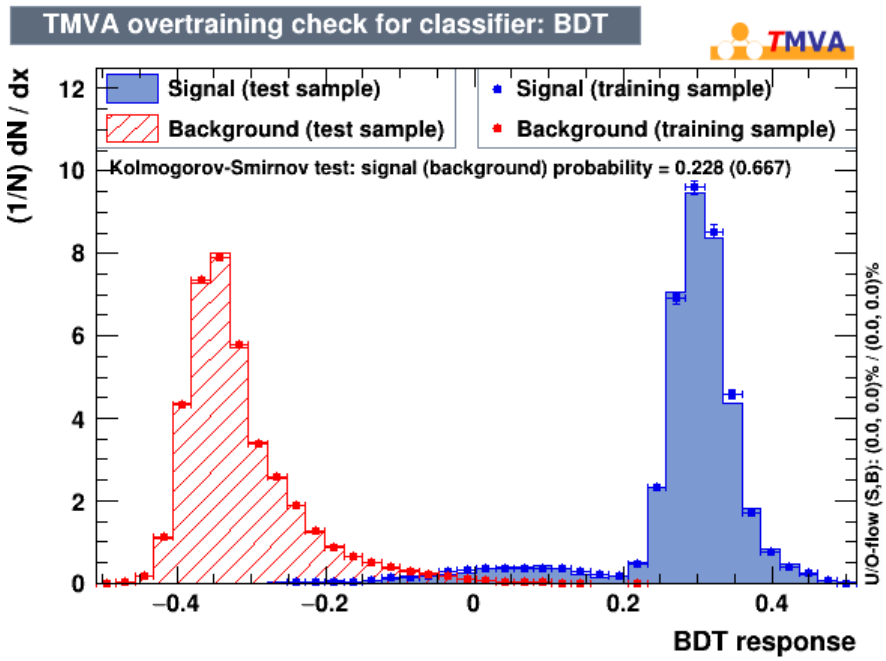

Figure 7.3: Classifier-BDT output distribution for signal and background (Testing and training events).

pass the BDT cut are classified as background(ER in this case). This separation can be clearly seen in the ER-NR band obtained in Figure 7.4(left panel) after the application of BDT cut. Green color represents events with BDT response > BDT cut value. Red color represents events with BDT response < BDT cut value. Further, we apply FV cut on the events classified by ML to remove the events which may lie near the surface or edges. This gives a more clear separation of the "bulk" events in the detector as shown in Figure 7.4(right panel) which shows the enhanced NR band after FV cut. Similarly, Figure 7.5(left panel) shows the ER-NR yield band separation by ML and Figure 7.5(right panel) shows the same after FV cut. The passage fraction (ratio of the number of events passing the FV cut and the total number of events) of the FV cut is found to be 67.7 %. The fraction of events classified as NR by ML improves from 54.49 % to 56.28 % after the application of the FV and the BDT cut.

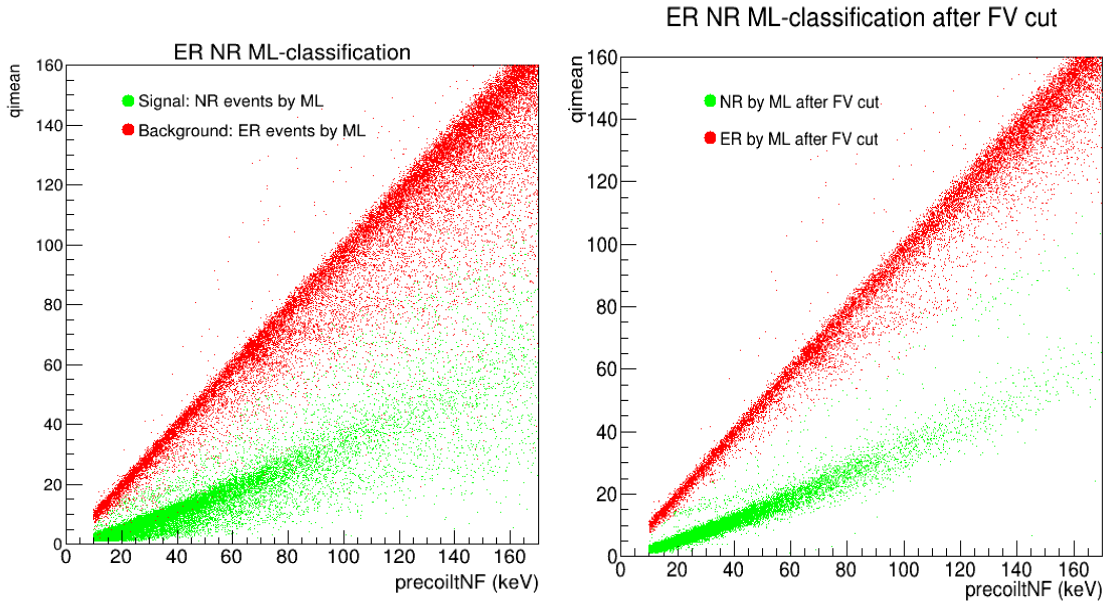

Figure 7.4: ER-NR bands separation after ML application. The x-axis is phonon energy. The y-axis is qimean which represents average of inner charge from Side 1 and Side 2.

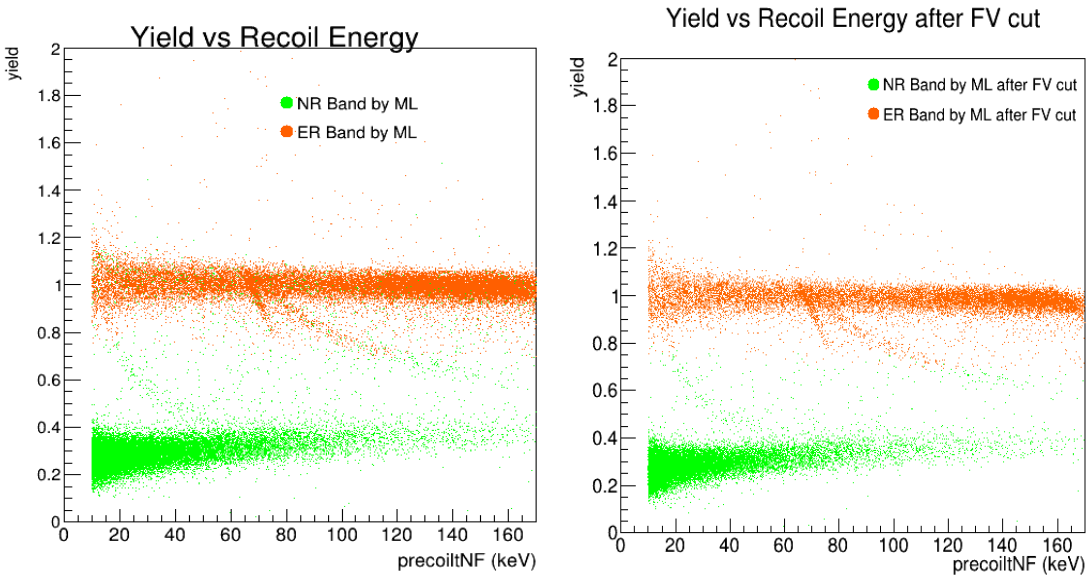

Figure 7.5: Yield vs. Recoil energy plot after ML application.

Chapter 8

Summary and Conclusions (SuperCDMS)

In this analysis, we have extracted Californium and Barium data for Nuclear recoils and Electron recoils, respectively. We have constructed the WIMP model by reweighting the Cf distribution, which gives the distribution spectra for WIMPs. We have constructed the Fiducial Volume cuts which define a region of "bulk" events in the detector. Finally, we have trained our ML BDT algorithm with NR events taken from Cf data and ER events taken from Ba data. The ML algorithm then classifies ER and NR events by applying the BDT cut criteria.

In conclusion, BDT cut provides an event by event separation of Electron Recoils and Nuclear Recoils. Finally, we see that FV cut removes unwanted surface and edge events and optimizes the Fiducial Volume of the detector and further enhances the NR band obtained from BDT. These NR band events obtained after the BDT cut and the FV cuts can be possible candidates for Dark Matter.

8.1 Outlook

We are now able to separate the NR events after the FV cuts and the BDT cut. We can try to achieve further classification to pick out the NR events which may have possibly come from WIMPs and separate them from the NR events which were generated by neutrons.

References

[1] B. Abelev, J. Adam, D. Adamová, M. M. Aggarwal, M. Agnello, A. Agostinelli, N. Agrawal, Z. Ahammed, N. Ahmad, A. A. Masoodi, *et al.*, "K*(892) 0 and ϕ (1020) production in pb-pb collisions at s n n= 2.76 tev," *Physical Review C*, vol. 91, no. 2, p. 024609, 2015.

[2] K. Kourou, T. P. Exarchos, K. P. Exarchos, M. V. Karamouzis, and D. I. Fotiadis, "Machine learning applications in cancer prognosis and prediction," *Computational and Structural Biotechnology Journal*, vol. 13, pp. 8 – 17, 2015.

[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *Journal of machine learning research*, vol. 12, no. Aug, pp. 2493–2537, 2011.

[4] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[5] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, pp. 3–24, 2007.

[6] CERN-ROOT, "Root user's guide." https://root.cern.ch/root/html534/guides/users-guide/ROOTUsersGuideA4.pdf, 2013. [Online; accessed 21-April-2019].

[7] A. Hoecker, K. Voss, H. Voss, A. Krasznahorkay, A. Christov, S. Henrot-Versillé, Y. Mahalalel, J. Stelzer, P. Speckmayer, M. Jachowski, *et al.*, "Tmva-toolkit for multivariate data analysis with root: Users guide," tech. rep., 2007.

[8] Y. Freund, R. Schapire, and N. Abe, "A short introduction to boosting," *Journal-Japanese Society For Artificial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[9] D. Ciupke, "Study of bdt training configurations with an application to the z/h $\tau\tau$ ee analysis," *University of Göttingen. Germany*, vol. 6, 2012.

[10] T. Sjöstrand, S. Mrenna, and P. Skands, "A brief introduction to pythia 8.1," *Computer Physics Communications*, vol. 178, no. 11, pp. 852–867, 2008.

[11] B. Abelev, J. Adam, D. Adamova, A. Adare, M. Aggarwal, G. A. Rinella, A. Agocs, A. Agostinelli, S. A. Salazar, Z. Ahammed, *et al.*, "Production of k*(892) 0 and ϕ (1020) in pp collisions at s n n= 7 tev," *The European Physical Journal C*, vol. 72, no. 10, p. 2183, 2012.

[12] A. Collaboration, "Performance of the alice experiment at the cern lhc," *International Journal of Modern Physics A*, vol. 29, no. 24, p. 1430044, 2014.

[13] L. Bergström, "Dark matter evidence, particle physics candidates and detection methods," *Annalen der Physik*, vol. 524, no. 9-10, pp. 479–496, 2012.

[14] B. D. Sherwin, J. Dunkley, S. Das, J. W. Appel, J. R. Bond, C. S. Carvalho, M. J. Devlin, R. Dünner, T. Essinger-Hileman, J. W. Fowler, *et al.*, "Evidence for dark energy from the cosmic microwave background alone using the atacama cosmology telescope lensing measurements," *Physical Review Letters*, vol. 107, no. 2, p. 021302, 2011.

[15] P. A. Ade, N. Aghanim, M. Arnaud, M. Ashdown, J. Aumont, C. Baccigalupi, A. Banday, R. Barreiro, J. Bartlett, N. Bartolo, *et al.*, "Planck 2015 results-xiii. cosmological parameters," *Astronomy & Astrophysics*, vol. 594, p. A13, 2016.

[16] G. Bertone, D. Hooper, and J. Silk, "Particle dark matter: Evidence, candidates and constraints," *Physics reports*, vol. 405, no. 5-6, pp. 279–390, 2005.

[17] V. C. Rubin, W. K. Ford Jr, and N. Thonnard, "Rotational properties of 21 sc galaxies with a large range of luminosities and radii, from ngc 4605/r= 4kpc/to ugc 2885/r= 122 kpc," *The Astrophysical Journal*, vol. 238, pp. 471–487, 1980.

[18] D. Clowe, M. Bradač, A. H. Gonzalez, M. Markevitch, S. W. Randall, C. Jones, and D. Zaritsky, "A direct empirical proof of the existence of dark matter," *The Astrophysical Journal Letters*, vol. 648, no. 2, p. L109, 2006.

[19] P. A. Ade, N. Aghanim, M. Arnaud, M. Ashdown, J. Aumont, C. Baccigalupi, A. Banday, R. Barreiro, J. Bartlett, N. Bartolo, *et al.*, "Planck 2015 results-xiii. cosmological parameters," *Astronomy & Astrophysics*, vol. 594, p. A13, 2016.

[20] L. Bergström, "Dark matter evidence, particle physics candidates and detection methods," *Annalen der Physik*, vol. 524, no. 9-10, pp. 479–496, 2012.

[21] P. Brink, B. Cabrera, J. Castle, J. Cooley, L. Novak, R. Ogburn, M. Pyle, J. Ruderman, A. Tomada, B. Young, *et al.*, "First test runs of a dark-matter detector with interleaved ionization electrodes and phonon sensors for surface-event rejection," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 559, no. 2, pp. 414–416, 2006.

[22] M. Pyle, B. Serfass, P. Brink, B. Cabrera, M. Cherry, N. Mirabolfathi, L. Novak, B. Sadoulet, D. Seitz, K. Sundqvist, *et al.*, "Surface electron rejection from ge detector with interleaved charge and phonon channels," in *AIP Conference Proceedings*, vol. 1185, pp. 223–226, AIP, 2009.

[23] R. Agnese, A. J. Anderson, D. Balakishiyeva, R. Basu Thakur, D. A. Bauer, A. Borgland, D. Brandt, P. L. Brink, R. Bunker, B. Cabrera, *et al.*, "Demonstration of surface electron rejection with interleaved germanium detectors for dark matter searches," *Applied Physics Letters*, vol. 103, no. 16, p. 164105, 2013.

[24] J. Lewin and P. Smith, "Review of mathematics, numerical factors, and corrections for dark matter experiments based on elastic nuclear recoil," *Astroparticle Physics*, vol. 6, no. 1, pp. 87–112, 1996.

[25] R. Agnese, T. Aramaki, I. Arnquist, W. Baker, D. Balakishiyeva, S. Banik, D. Barker, R. B. Thakur, D. Bauer, T. Binder, *et al.*, "Results from the super cryogenic dark matter search experiment at soudan," *Physical review letters*, vol. 120, no. 6, p. 061802, 2018.

Appendix A

8.1 Formula

Let S_f: Truely identified signal events and B_f: Truely identified background events.

Let S_o be the total signal and B_o be the total background.

- Signal efficiency : S_f/S_o
- Background rejection : $r_B = B_f/B_o$
- Background efficiency : $1 - r_B$

8.2 SuperCDMS variables

- ptNF : Phonon total pulse energy from non-stationary optimal filter.
- plukeqOF : Estimate of luke phonons, using only the charge signal (OF: Optimal Filter).
- precoiltNF : The non-luke phonon energy (ptNF-plukeqOF).
- qsum#OF : Charge sum of side #=1,2 i.e. qi#OF + qo#OF.
- qimean : Average of inner charge i.e. (qi1OF + qi2OF)/2.
- qsummaxOF : Sum for the side with the maximum qsum i.e. max(qsum#OF,qsum#OF).
- ytNF: Yield i.e qsummaxOF/precoiltNF.

- qzpartOF: Primary face fiducialization parameter, charge z partition i.e. (qsum1OF-qsum2OF)/(qsum1OF+qsum2OF).
- qrpart1OF: Primary sidewall fiducialization parameter i.e. qo1OF/qsum1OF.
- pzpartOF: Bulk z position proxy i.e. (pt1OF-pt2OF)/(pt1OF+pt2OF).
- prpartOF: Bulk r position proxy i.e. (po1OF+po2OF)/psumOF.

8.3 Scripts (ALICE)

8.3.1 Making signal and background trees for K^{*0}.

```
#define Kstar_cxx
#include "Kstar.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TFile.h>
#include<math.h>
#include"TROOT.h"
#include<iostream>

void Kstar::Loop()
{
//   In a ROOT session, you can do:
//      root> .L Kstar.C
//      root> Kstar t
//      root> t.GetEntry(12); // Fill t data members with entry number 12
//      root> t.Show();       // Show values of entry 12
//      root> t.Show(16);     // Read and show values of entry 16
//      root> t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0); // disable all branches
```

```
//     fChain->SetBranchStatus("branchname",1); // activate branchname
// METHOD2: replace line
//     fChain->GetEntry(jentry);       //read all  branches
//by  b_branchname->GetEntry(ientry); //read only this branch
// Mass of K+ =493.677 MeV/c2 .....Mass of 139.57061 MeV/c2 ... Mass of Kstar0=895.55
    Mev/c2
int   Event=0;


float  invmass_s,d1pt_s,d2pt_s,costheta_s,motherpt_s,
dcazd1_s,dcaxyd1_s,dcazd2_s,dcaxyd2_s,dcad1_s,dcad2_s,motherp_s,
    d1p_s,d2p_s,d1eta_s,d2eta_s,mothereta_s,motherE_s,costs_s,costs1_s;
float  invmass_b,d1pt_b,d2pt_b,costheta_b,motherpt_b,
dcazd1_b,dcaxyd1_b,dcazd2_b,dcaxyd2_b,dcad1_b,dcad2_b,motherp_b,
    d1p_b,d2p_b,d1eta_b,d2eta_b,mothereta_b,motherE_b,costs_b,costs1_b;


// TFile *f1 = new TFile("application2_read.root","RECREATE");
TFile *f1 = new TFile("likesignplusplusfeatures_0.8pt1.2.root","RECREATE");
TTree *tree_sig = new TTree("tree_sig","Signal tree");
TTree *tree_bkg = new TTree("tree_bkg","Background tree");


tree_sig->Branch("invmass",&invmass_s,"invmass_s/F");
tree_sig->Branch("daughter1pt",&d1pt_s,"d1pt_s/F");
tree_sig->Branch("daughter2pt",&d2pt_s,"d2pt_s/F");
tree_sig->Branch("costheta",&costheta_s,"costheta_s/F");
tree_sig->Branch("motherpt",&motherpt_s,"motherpt_s/F");
tree_sig->Branch("dcazd1",&dcazd1_s,"dcazd1_s/F");
tree_sig->Branch("dcazd2",&dcazd2_s,"dcazd2_s/F");
tree_sig->Branch("dcaxyd1",&dcaxyd1_s,"dcaxyd1_s/F");
tree_sig->Branch("dcaxyd2",&dcaxyd2_s,"dcaxyd2_s/F");
tree_sig->Branch("dcad1",&dcad1_s,"dcad1_s/F");
tree_sig->Branch("dcad2",&dcad2_s,"dcad2_s/F");
tree_sig->Branch("motherp",&motherp_s,"motherp_s/F");
tree_sig->Branch("daughter1p",&d1p_s,"d1p_s/F");
tree_sig->Branch("daughter2p",&d2p_s,"d2p_s/F");
tree_sig->Branch("daughter1eta",&d1eta_s,"d1eta_s/F");
tree_sig->Branch("daughter2eta",&d2eta_s,"d2eta_s/F");
tree_sig->Branch("mothereta",&mothereta_s,"mothereta_s/F");
tree_sig->Branch("motherE",&motherE_s,"motherE_s/F");
tree_sig->Branch("costs",&costs_s,"costs_s/F");
tree_sig->Branch("costs1",&costs1_s,"costs1_s/F");


tree_bkg->Branch("invmass",&invmass_b,"invmass_b/F");
tree_bkg->Branch("daughter1pt",&d1pt_b,"d1pt_b/F");
tree_bkg->Branch("daughter2pt",&d2pt_b,"d2pt_b/F");
tree_bkg->Branch("costheta",&costheta_b,"costheta_b/F");
tree_bkg->Branch("motherpt",&motherpt_b,"motherpt_b/F");
tree_bkg->Branch("dcazd1",&dcazd1_b,"dcazd1_b/F");
```

```
tree_bkg->Branch("dcazd2",&dcazd2_b,"dcazd2_b/F");
tree_bkg->Branch("dcaxyd1",&dcaxyd1_b,"dcaxyd1_b/F");
tree_bkg->Branch("dcaxyd2",&dcaxyd2_b,"dcaxyd2_b/F");
tree_bkg->Branch("dcad1",&dcad1_b,"dcad1_b/F");
tree_bkg->Branch("dcad2",&dcad2_b,"dcad2_b/F");
tree_bkg->Branch("motherp",&motherp_b,"motherp_b/F");
tree_bkg->Branch("daughter1p",&d1p_b,"d1p_b/F");
tree_bkg->Branch("daughter2p",&d2p_b,"d2p_b/F");
tree_bkg->Branch("daughter1eta",&d1eta_b,"d1eta_b/F");
tree_bkg->Branch("daughter2eta",&d2eta_b,"d2eta_b/F");
tree_bkg->Branch("mothereta",&mothereta_b,"mothereta_b/F");
tree_bkg->Branch("motherE",&motherE_b,"motherE_b/F");
tree_bkg->Branch("costs",&costs_b,"costs_b/F");
tree_bkg->Branch("costs1",&costs1_b,"costs1_b/F");

const float massK=0.493677; //GeV
const float massPi=0.13957061; //GeV

int k1,k2;
float mass,Energyd1,Energyd2,cosangle;
float mass_b,Energyd1_b,Energyd2_b,cosangle_b;
TLorentzVector trackm(0,0,0,0);
TLorentzVector trackd1(0,0,0,0); TLorentzVector trackd1_clone(0,0,0,0);
TLorentzVector trackd2(0,0,0,0);

TVector3 vectord1;
TVector3 vectord2;
TVector3 vectormother,vectormother_b;

TLorentzVector trackm_b(0,0,0,0);
TLorentzVector trackd1_b(0,0,0,0);
TLorentzVector trackd2_b(0,0,0,0);

TVector3 momentumD(0,0,0);
TVector3 momentumM(0,0,0);

TVector3 vectord1_b;
TVector3 vectord2_b;

//TH1D *hsig=new TH1D("hsig","Invariant Mass distribution of K*^{0}",90,0.6,1.5);
//TH1D *hcos=new TH1D("hcos","cos #theta between momentum 3 vector of daughters K^{+}
    and #pi^{-} ",200,-1,1);

if (fChain == 0) return;

Long64_t nentries = fChain->GetEntriesFast();


cout<<"Number of events
    are----------------------------------------"<<nentries<<endl;
```

```
//********EVENT LOOP*****************************************
Long64_t nbytes = 0, nb = 0;
for (Long64_t jentry=0; jentry<nentries;jentry++)
{
int sum_s=0.0;
int sum_b=0.0;

Event=jentry;


Long64_t ientry = LoadTree(jentry);
if (ientry < 0) break;
nb = fChain->GetEntry(jentry); nbytes += nb;
// if (Cut(ientry) < 0) continue;
//cout<<"EVENT
    number------------------------------">>jentry<<endl;

if (jentry%20000==0)cout<<"Number of events processed ==========
    ">>jentry<<endl;

if (TMath::Abs(fEvSel_VtxZ)<10)
{
// Track loop*****************************************************//

for (k1=0;k1<fTreeEventNTrack;k1++)
{
//cout<<"Track ">>k<<endl;
if (TMath::Abs(fTreeTrackVariableEta[k1])<0.8 && fTreeTrackVariablePt[k1]>0.15 &&
    fTreeTrackVariableTpcNCrossedRows[k1]>70 &&
    fTreeTrackVariableLeastRatioCrossedRowsOverFindable[k1]>0.8
&& fTreeTrackVariableChiSqrPerTpcCls[k1]<4 && fTreeTrackVariableChiSqrPerItsCls[k1]<36
    && TMath::Abs(fTreeTrackVariableDcaZ[k1])<2
&& TMath::Abs(fTreeTrackVariableDcaXY[k1])< 7*(0.0026 + 0.005/fTreeTrackVariablePt[k1]) )
{
//cout<<"Conditions fulfilled  ">>endl;

for (k2=0;k2<fTreeEventNTrack;k2++)
{
if (TMath::Abs(fTreeTrackVariableEta[k2])<0.8 && fTreeTrackVariablePt[k2]>0.15 &&
    fTreeTrackVariableTpcNCrossedRows[k2]>70 &&
    fTreeTrackVariableLeastRatioCrossedRowsOverFindable[k2]>0.8
&& fTreeTrackVariableChiSqrPerTpcCls[k2]<4 && fTreeTrackVariableChiSqrPerItsCls[k2]<36
    && TMath::Abs(fTreeTrackVariableDcaZ[k2])<2
&& TMath::Abs(fTreeTrackVariableDcaXY[k2])< 7*(0.0026 + 0.005/fTreeTrackVariablePt[k2]) )
{
//-----------------SIGNAL--------------------------
if (fTreeMCPid[k1]==321 && fTreeMCPid[k2]==-211) // K+ =321 and Pi- =-211 ; Kstar0
    =313
{
//cout<<"We got K+ and Pi- pairs " <<endl;
```

```
if (fTreeMCMotherTrId[k1]==fTreeMCMotherTrId[k2] && fTreeMCMotherPid[k1]==313 &&
    fTreeMCMotherPid[k2]==313)
{
// cout<<"It is K star0     !!!!
//----------------------------in the Event number="<<jentry<<endl;
// cout<<"Track mother from 1="<<setw(5)<<fTreeMCMotherTrId[k1]<<
//" Track Mother from 2="<<setw(5)<<fTreeMCMotherTrId[k2]<<endl;
//cout<<"PID from 1="<<setw(5)<<fTreeMCMotherPid[k1]<<" PID from
    2="<<setw(5)<<fTreeMCMotherPid[k2]<<endl;
// Mass of K+ =493.677 MeV/c2 .....Mass of Pi =139.57061 MeV/c2
vectord1.SetXYZ(fTreeTrackVariableMomentumPx[k1],fTreeTrackVariableMomentumPy[k1],
fTreeTrackVariableMomentumPz[k1]);
vectord2.SetXYZ(fTreeTrackVariableMomentumPx[k2],fTreeTrackVariableMomentumPy[k2],
fTreeTrackVariableMomentumPz[k2]);

Energyd1=TMath::Sqrt(massK*massK + vectord1.Mag2() );
Energyd2=TMath::Sqrt(massPi*massPi + vectord2.Mag2() );

trackd1.SetPxPyPzE(fTreeTrackVariableMomentumPx[k1],fTreeTrackVariableMomentumPy[k1],
fTreeTrackVariableMomentumPz[k1],Energyd1);
trackd2.SetPxPyPzE(fTreeTrackVariableMomentumPx[k2],fTreeTrackVariableMomentumPy[k2],
fTreeTrackVariableMomentumPz[k2],Energyd2);
trackm=trackd1+trackd2;
vectormother.SetXYZ(trackm.Px(),trackm.Py(),trackm.Pz());
trackd1_clone=trackd1;
if (/*TMath::Abs(trackm.Rapidity())<0.5 &&*/ 0.6<trackm.Mag() && trackm.Mag() <1.2 &&
    trackm.Pt()>=0.8 && trackm.Pt()<=1.2)
{

motherp_s=vectormother.Mag();
d1p_s=vectord1.Mag();
d2p_s=vectord2.Mag();


mass=trackm.Mag();
invmass_s=mass;
//hsig->Fill(mass);
d1pt_s=trackd1.Pt();
d2pt_s=trackd2.Pt();
motherpt_s=trackm.Pt();


cosangle=(vectord1*vectord2)/(vectord1.Mag()*vectord2.Mag());
costheta_s=cosangle;
//hcos->Fill(cosangle);

dcazd1_s=fTreeTrackVariableDcaZ[k1];
dcazd2_s=fTreeTrackVariableDcaZ[k2];
// cout<<"Difference dca z d1-d2 "<<dcazd1_s-dcazd2_s<<endl;
dcaxyd1_s=fTreeTrackVariableDcaXY[k1];
dcaxyd2_s=fTreeTrackVariableDcaXY[k2];
```

```
//   cout<<"Difference dca xy d1-d2 "<<dcaxyd1_s-dcaxyd2_s<<endl;
dcad1_s=TMath::Sqrt(dcazd1_s*dcazd1_s + dcaxyd1_s*dcaxyd1_s);
dcad2_s=TMath::Sqrt(dcazd2_s*dcazd2_s + dcaxyd2_s*dcaxyd2_s);
// cout<<"Difference dca d1-d2  "<<dcad1_s-dcad2_s<<endl;
d1eta_s=fTreeTrackVariableEta[k1];
d2eta_s=fTreeTrackVariableEta[k2];
mothereta_s=trackm.Eta();
motherE_s=trackm.E();

momentumM=trackm.Vect();
TVector3 normal(trackm.Y() / trackm.Pt(), -trackm.X() / trackm.Pt(), 0.0);
TVector3 normal1(0,0,1);
Double_t betaX = -trackm.X() / trackm.E();
Double_t betaY = -trackm.Y() / trackm.E();
Double_t betaZ = -trackm.Z() / trackm.E();
trackd1_clone.Boost(betaX, betaY, betaZ);
momentumD = trackd1_clone.Vect();
Double_t cosThetaStar = TMath::Abs(normal.Dot(momentumD)) / momentumD.Mag();
Double_t cosThetaStar1 = TMath::Abs(normal1.Dot(momentumD)) / momentumD.Mag();
costs_s=cosThetaStar;
costs1_s=cosThetaStar1;

tree_sig->Fill();

}//Rapidity cut
}
} //Pid selection  if  loop ends  // Signal Loop


// Like sign background
if(fTreeMCPid[k1]==321 && fTreeMCPid[k2]==211) // K+ =321 and Pi =211
{

//   cout<<"Like sign background----------------------------"<<endl;
vectord1_b.SetXYZ(fTreeTrackVariableMomentumPx[k1],fTreeTrackVariableMomentumPy[k1],
fTreeTrackVariableMomentumPz[k1]);
vectord2_b.SetXYZ(fTreeTrackVariableMomentumPx[k2],fTreeTrackVariableMomentumPy[k2],
fTreeTrackVariableMomentumPz[k2]);

Energyd1_b=TMath::Sqrt(massK*massK + vectord1_b.Mag2() );
Energyd2_b=TMath::Sqrt(massPi*massPi + vectord2_b.Mag2() );

trackd1_b.SetPxPyPzE(fTreeTrackVariableMomentumPx[k1],fTreeTrackVariableMomentumPy[k1],
fTreeTrackVariableMomentumPz[k1],Energyd1_b);
trackd2_b.SetPxPyPzE(fTreeTrackVariableMomentumPx[k2],fTreeTrackVariableMomentumPy[k2],
fTreeTrackVariableMomentumPz[k2],Energyd2_b);
trackm_b=trackd1_b+trackd2_b;
vectormother_b.SetXYZ(trackm_b.Px(),trackm_b.Py(),trackm_b.Pz());
trackd1_clone=trackd1_b;
if(/*TMath::Abs(trackm_b.Rapidity())<0.5 &&*/ 0.6< trackm_b.Mag() &&
    trackm_b.Mag()<1.2 && trackm_b.Pt()>=0.8 && trackm_b.Pt()<=1.2)
```

```
{



motherp_b=vectormother_b.Mag();
d1p_b=vectord1_b.Mag();
d2p_b=vectord2_b.Mag();
mass_b=trackm_b.Mag();
invmass_b=mass_b;
//hsig->Fill(mass);
d1pt_b=trackd1_b.Pt();
d2pt_b=trackd2_b.Pt();
motherpt_b=trackm_b.Pt();


cosangle_b=(vectord1_b*vectord2_b)/(vectord1_b.Mag()*vectord2_b.Mag());
costheta_b=cosangle_b;
//hcos->Fill(cosangle);

dcazd1_b=fTreeTrackVariableDcaZ[k1];
dcazd2_b=fTreeTrackVariableDcaZ[k2];
// cout<<"Difference bkg dca z d1-d2  "<<dcazd1_b-dcazd2_b<<endl;
dcaxyd1_b=fTreeTrackVariableDcaXY[k1];
dcaxyd2_b=fTreeTrackVariableDcaXY[k2];
//cout<<"Difference dca bkg xy d1-d2 "<<dcaxyd1_b-dcaxyd2_b<<endl;
dcad1_b=TMath::Sqrt(dcazd1_b*dcazd1_b + dcaxyd1_b*dcaxyd1_b);
dcad2_b=TMath::Sqrt(dcazd2_b*dcazd2_b + dcaxyd2_b*dcaxyd2_b);
// cout<<"Difference dca bkg d1-d2 "<<dcad1_b-dcad2_b<<endl;
d1eta_b=fTreeTrackVariableEta[k1];
d2eta_b=fTreeTrackVariableEta[k2];
mothereta_b=trackm_b.Eta();
motherE_b=trackm_b.E();

momentumM=trackm_b.Vect();
TVector3 normal(trackm_b.Y() / trackm_b.Pt(), -trackm_b.X() / trackm_b.Pt(), 0.0);
TVector3 normal1(0,0,1);
Double_t betaX = -trackm_b.X() / trackm_b.E();
Double_t betaY = -trackm_b.Y() / trackm_b.E();
Double_t betaZ = -trackm_b.Z() / trackm_b.E();
trackd1_clone.Boost(betaX, betaY, betaZ);
momentumD = trackd1_clone.Vect();
Double_t cosThetaStar = TMath::Abs(normal.Dot(momentumD)) / momentumD.Mag();
Double_t cosThetaStar1 = TMath::Abs(normal1.Dot(momentumD)) / momentumD.Mag();
costs_b=cosThetaStar;
costs1_b=cosThetaStar1;

tree_bkg->Fill();
} //Rapidity cut
} // Like sign ends ++----------------

/*   if(fTreeMCPid[k1]==-321 && fTreeMCPid[k2]==-211) // K+ =321 and Pi =211
```

```
{
//   cout<<"Like sign background----------------------------------"<<endl;
vectord1_b.SetXYZ(fTreeTrackVariableMomentumPx[k1],fTreeTrackVariableMomentumPy[k1],
fTreeTrackVariableMomentumPz[k1]);
vectord2_b.SetXYZ(fTreeTrackVariableMomentumPx[k2],fTreeTrackVariableMomentumPy[k2],
fTreeTrackVariableMomentumPz[k2]);

Energyd1_b=TMath::Sqrt(massK*massK + vectord1_b.Mag2() );
Energyd2_b=TMath::Sqrt(massPi*massPi + vectord2_b.Mag2() );

trackd1_b.SetPxPyPzE(fTreeTrackVariableMomentumPx[k1],fTreeTrackVariableMomentumPy[k1],
fTreeTrackVariableMomentumPz[k1],Energyd1_b);
trackd2_b.SetPxPyPzE(fTreeTrackVariableMomentumPx[k2],fTreeTrackVariableMomentumPy[k2],
fTreeTrackVariableMomentumPz[k2],Energyd2_b);
trackm_b=trackd1_b+trackd2_b;
vectormother_b.SetXYZ(trackm_b.Px(),trackm_b.Py(),trackm_b.Pz());
 if (TMath::Abs(trackm_b.Rapidity())<0.5 && 0.6< trackm_b.Mag() && trackm_b.Mag()<1.2)
{
motherp_b=vectormother_b.Mag();
d1p_b=vectord1_b.Mag();
d2p_b=vectord2_b.Mag();
mass_b=trackm_b.Mag();
invmass_b=mass_b;
//hsig->Fill(mass);
d1pt_b=trackd1_b.Pt();
d2pt_b=trackd2_b.Pt();
motherpt_b=trackm_b.Pt();

cosangle_b=(vectord1_b*vectord2_b)/(vectord1_b.Mag()*vectord2_b.Mag());
costheta_b=cosangle_b;
//hcos->Fill(cosangle);
dcazd1_b=fTreeTrackVariableDcaZ[k1];
dcazd2_b=fTreeTrackVariableDcaZ[k2];
// cout<<"Difference bkg dca z d1-d2  "<<dcazd1_b-dcazd2_b<<endl;
dcaxyd1_b=fTreeTrackVariableDcaXY[k1];
dcaxyd2_b=fTreeTrackVariableDcaXY[k2];
//cout<<"Difference dca bkg xy d1-d2 "<<dcaxyd1_b-dcaxyd2_b<<endl;
dcad1_b=TMath::Sqrt(dcazd1_b*dcazd1_b + dcaxyd1_b*dcaxyd1_b);
dcad2_b=TMath::Sqrt(dcazd2_b*dcazd2_b + dcaxyd2_b*dcaxyd2_b);
// cout<<"Difference dca bkg d1-d2 "<<dcad1_b-dcad2_b<<endl;
tree_bkg->Fill();
} //Rapidity cut
} // Like sign ends --   ------------------*/

} //selection conditions for track2

} //k2 track for loop ends


} // selection conditions if ends
```

```
}// **************end of TRACK1 loop*************************


} // Event selection  conditions

} // **********************End of Event loop************************

//hsig->Draw();
// hcos->Draw();

f1->cd();
 tree_sig ->Write();
tree_bkg->Write();
f1->Close();
}
```

TMVA Classification code for K^{*0}.

```
/// \ file
/// \ingroup tutorial_tmva
/// \notebook -nodraw
/// This macro provides a simple example on how to use the trained  classifiers
/// within an analysis  module
/// - Project   : TMVA - a Root-integrated toolkit for multivariate data analysis
/// - Package   : TMVA
/// - Exectuable: TMVAClassificationApplication
///
/// \macro_output
/// \macro_code
/// \author Andreas Hoecker

/*
root -l ./TMVAClassificationApplication_kstar.C\(\"BDT\"\)
*/
#include <cstdlib>
#include <vector>
#include <iostream>
#include <map>
#include <string>

#include "TFile.h"
#include "TTree.h"
#include "TString.h"
#include "TSystem.h"
#include "TROOT.h"
#include "TStopwatch.h"

#include "TMVA/Tools.h"
```

```
#include "TMVA/Reader.h"
#include "TMVA/MethodCuts.h"

using namespace TMVA;

void TMVAClassificationApplication_kstar( TString myMethodList = "" )
{

double cutvalue=-0.0932; //like
cout<<"THE CUT VALUE APPLIED
    IS======================="<<cutvalue<<endl;


//---------------------------------------------------
// This loads the library
TMVA::Tools::Instance();

// Default MVA methods to be trained + tested
std::map<std::string,int> Use;

// Cut optimisation
Use["Cuts"]            = 1;
Use["CutsD"]           = 1;
Use["CutsPCA"]         = 0;
Use["CutsGA"]          = 0;
Use["CutsSA"]          = 0;
//
// 1-dimensional likelihood ("naive Bayes estimator")
Use["Likelihood"]      = 1;
Use["LikelihoodD"]     = 0; // the "D" extension indicates decorrelated input variables (see
    option strings)
Use["LikelihoodPCA"]   = 1; // the "PCA" extension indicates PCA-transformed input variables
    (see option strings)
Use["LikelihoodKDE"]   = 0;
Use["LikelihoodMIX"]   = 0;
//
// Mutidimensional likelihood and Nearest-Neighbour methods
Use["PDERS"]           = 1;
Use["PDERSD"]          = 0;
Use["PDERSPCA"]        = 0;
Use["PDEFoam"]         = 1;
Use["PDEFoamBoost"]    = 0; // uses generalised MVA method boosting
Use["KNN"]             = 1; // k-nearest neighbour method
//
// Linear Discriminant Analysis
Use["LD"]              = 1; // Linear Discriminant identical to Fisher
Use["Fisher"]          = 0;
Use["FisherG"]         = 0;
Use["BoostedFisher"]   = 0; // uses generalised MVA method boosting
Use["HMatrix"]         = 0;
//
```

```
// Function Discriminant analysis
Use["FDA_GA"]          = 1; // minimisation of user-defined function using Genetics Algorithm
Use["FDA_SA"]          = 0;
Use["FDA_MC"]          = 0;
Use["FDA_MT"]          = 0;
Use["FDA_GAMT"]        = 0;
Use["FDA_MCMT"]        = 0;
//
// Neural Networks (all are feed-forward Multilayer Perceptrons)
Use["MLP"]             = 0; // Recommended ANN
Use["MLPBFGS"]         = 0; // Recommended ANN with optional training method
Use["MLPBNN"]          = 1; // Recommended ANN with BFGS training method and bayesian
    regulator
Use["CFMlpANN"]        = 0; // Depreciated ANN from ALEPH
Use["TMlpANN"]         = 0; // ROOT's own ANN
Use["DNN"]             = 0; // improved implementation of a NN
//
// Support Vector Machine
Use["SVM"]             = 1;
//
// Boosted Decision Trees
Use["BDT"]             = 1; // uses Adaptive Boost
Use["BDTG"]            = 0; // uses Gradient Boost
Use["BDTB"]            = 0; // uses Bagging
Use["BDTD"]            = 0; // decorrelation + Adaptive Boost
Use["BDTF"]            = 0; // allow usage of fisher discriminant for node splitting
//
// Friedman's RuleFit method, ie, an optimised series of cuts ("rules")
Use["RuleFit"]         = 1;
// ---------------------------------
Use["Plugin"]          = 0;
Use["Category"]        = 0;
Use["SVM_Gauss"]       = 0;
Use["SVM_Poly"]        = 0;
Use["SVM_Lin"]         = 0;

std::cout << std::endl;
std::cout << "==> Start TMVAClassificationApplication" << std::endl;

// Select methods (don't look at this code - not of interest)
if (myMethodList != "") {
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) it->second
    = 0;

std::vector<TString> mlist = gTools().SplitString( myMethodList, ',' );
for (UInt_t i=0; i<mlist.size(); i++) {
std::string regMethod(mlist[i]);

if (Use.find(regMethod) == Use.end()) {
std::cout << "Method \"" << regMethod
<< "\" not known in TMVA under this name. Choose among the following:" << std::endl;
```

```
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) {
std::cout << it->first << " ";
}
std::cout << std::endl;
return;
}
Use[regMethod] = 1;
}
}

// ---------------------------------------------------------------

// Create the Reader object

TMVA::Reader *reader = new TMVA::Reader( "!Color:!Silent" );

// Create a set of variables and declare them to the reader
// - the variable names MUST corresponds in name and type to those given in the weight file(s)
    used
/*
Float_t var1, var2;
Float_t var3, var4;
reader->AddVariable( "myvar1 := var1+var2", &var1 );
reader->AddVariable( "myvar2 := var1-var2", &var2 );
reader->AddVariable( "var3",                &var3 );
reader->AddVariable( "var4",                &var4 );
*/


float
    invmass_u,d1pt_u,d2pt_u,motherpt_u,etad1_u,etad2_u,etadm_u,costs_u,costs1_u,dcad1_u,dcad2_u;
// reader->AddVariable("invmass",&invmass_u);
// reader->AddVariable("daughter1pt",&d1pt_u);
//reader->AddVariable("daughter2pt",&d2pt_u);
reader->AddVariable("motherpt",&motherpt_u);
//reader->AddVariable("etad1",&etad1_u);
//reader->AddVariable("etad2",&etad2_u);
reader->AddVariable("etadm",&etadm_u);
reader->AddVariable("costs",&costs_u);
//reader->AddVariable("costs1",&costs1_u);
reader->AddVariable("dcad1",&dcad1_u);
//reader->AddVariable("dcad2",&dcad2_u);

// Spectator variables declared in the training have to be added to the reader, too
/*
Float_t spec1,spec2;
reader->AddSpectator( "spec1 := var1*2", &spec1 );
reader->AddSpectator( "spec2 := var1*3", &spec2 );
*/
/* Float_t Category_cat1, Category_cat2, Category_cat3;
if (Use["Category"]){
```

```
// Add artificial spectators for distinguishing categories
reader->AddSpectator( "Category_cat1 := var3<=0",          &Category_cat1 );
reader->AddSpectator( "Category_cat2 := (var3>0)&&(var4<0)", &Category_cat2 );
reader->AddSpectator( "Category_cat3 := (var3>0)&&(var4>=0)", &Category_cat3 );
}*/

// Book the MVA methods

TString dir    = "dataset/weights/";
TString prefix = "TMVAClassification";

// Book method(s)
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) {
if (it->second) {
TString methodName = TString(it->first) + TString(" method");
TString weightfile = dir + prefix + TString("_") + TString(it->first) +
    TString(".weights.xml");
reader->BookMVA( methodName, weightfile );
}
}

// Book output histograms
UInt_t nbin = 100;
TH1F *histLk(0), *histLkD(0), *histLkPCA(0), *histLkKDE(0), *histLkMIX(0), *histPD(0),
    *histPDD(0);
TH1F *histPDPCA(0), *histPDEFoam(0), *histPDEFoamErr(0), *histPDEFoamSig(0),
    *histKNN(0), *histHm(0);
TH1F *histFi(0), *histFiG(0), *histFiB(0), *histLD(0),
    *histNn(0),*histNnbfgs(0),*histNnbnn(0);
TH1F *histNnC(0), *histNnT(0), *histNdn(0), *histBdt(0), *histBdtG(0), *histBdtB(0),
    *histBdtD(0);
TH1F *histBdtF(0), *histRf(0), *histSVMG(0), *histSVMP(0), *histSVML(0),
    *histFDAMT(0), *histFDAGA(0);
TH1F *histCat(0), *histPBdt(0);

TH1D *hinvmass_signal=new TH1D("invsig","invsig",60,0.6,1.2);
TH1D *hinvmass_bkg=new TH1D("invbkg","invbkg",60,0.6,1.2);

if (Use["Likelihood"])     histLk     = new TH1F( "MVA_Likelihood", "MVA_Likelihood",
    nbin, -1, 1 );
if (Use["LikelihoodD"])    histLkD    = new TH1F( "MVA_LikelihoodD", "MVA_LikelihoodD",
    nbin, -1, 0.9999 );
if (Use["LikelihoodPCA"]) histLkPCA = new TH1F( "MVA_LikelihoodPCA",
    "MVA_LikelihoodPCA", nbin, -1, 1 );
if (Use["LikelihoodKDE"]) histLkKDE = new TH1F( "MVA_LikelihoodKDE",
    "MVA_LikelihoodKDE", nbin, -0.00001, 0.99999 );
if (Use["LikelihoodMIX"]) histLkMIX = new TH1F( "MVA_LikelihoodMIX",
    "MVA_LikelihoodMIX", nbin, 0, 1 );
if (Use["PDERS"])          histPD     = new TH1F( "MVA_PDERS",   "MVA_PDERS",
    nbin,   0,  1 );
```

```
if (Use["PDERSD"])      histPDD    = new TH1F( "MVA_PDERSD", "MVA_PDERSD",
    nbin,  0, 1 );
if (Use["PDERSPCA"])    histPDPCA  = new TH1F( "MVA_PDERSPCA",
    "MVA_PDERSPCA", nbin, 0, 1 );
if (Use["KNN"])         histKNN    = new TH1F( "MVA_KNN",     "MVA_KNN",
    nbin,  0, 1 );
if (Use["HMatrix"])     histHm     = new TH1F( "MVA_HMatrix", "MVA_HMatrix",
    nbin, -0.95, 1.55 );
if (Use["Fisher"])      histFi     = new TH1F( "MVA_Fisher",  "MVA_Fisher",
    nbin, -4, 4 );
if (Use["FisherG"])     histFiG    = new TH1F( "MVA_FisherG", "MVA_FisherG",
    nbin, -1, 1 );
if (Use["BoostedFisher"]) histFiB  = new TH1F( "MVA_BoostedFisher",
    "MVA_BoostedFisher", nbin, -2, 2 );
if (Use["LD"])          histLD     = new TH1F( "MVA_LD",      "MVA_LD",
    nbin, -2, 2 );
if (Use["MLP"])         histNn     = new TH1F( "MVA_MLP",     "MVA_MLP",
    nbin, -1.25, 1.5 );
if (Use["MLPBFGS"])     histNnbfgs = new TH1F( "MVA_MLPBFGS", "MVA_MLPBFGS",
    nbin, -1.25, 1.5 );
if (Use["MLPBNN"])      histNnbnn  = new TH1F( "MVA_MLPBNN", "MVA_MLPBNN",
    nbin, -1.25, 1.5 );
if (Use["CFMlpANN"])    histNnC    = new TH1F( "MVA_CFMlpANN", "MVA_CFMlpANN",
    nbin, 0, 1 );
if (Use["TMlpANN"])     histNnT    = new TH1F( "MVA_TMlpANN", "MVA_TMlpANN",
    nbin, -1.3, 1.3 );
if (Use["DNN"])         histNdn    = new TH1F( "MVA_DNN",     "MVA_DNN",
    nbin, -0.1, 1.1 );
if (Use["BDT"])         histBdt    = new TH1F( "MVA_BDT",     "MVA_BDT",
    nbin, -0.8, 0.8 );
if (Use["BDTG"])        histBdtG   = new TH1F( "MVA_BDTG",    "MVA_BDTG",
    nbin, -1.0, 1.0 );
if (Use["BDTB"])        histBdtB   = new TH1F( "MVA_BDTB",    "MVA_BDTB",
    nbin, -1.0, 1.0 );
if (Use["BDTD"])        histBdtD   = new TH1F( "MVA_BDTD",    "MVA_BDTD",
    nbin, -0.8, 0.8 );
if (Use["BDTF"])        histBdtF   = new TH1F( "MVA_BDTF",    "MVA_BDTF",
    nbin, -1.0, 1.0 );
if (Use["RuleFit"])     histRf     = new TH1F( "MVA_RuleFit", "MVA_RuleFit",
    nbin, -2.0, 2.0 );
if (Use["SVM_Gauss"])   histSVMG   = new TH1F( "MVA_SVM_Gauss", "MVA_SVM_Gauss",
    nbin, 0.0, 1.0 );
if (Use["SVM_Poly"])    histSVMP   = new TH1F( "MVA_SVM_Poly", "MVA_SVM_Poly",
    nbin,  0.0, 1.0 );
if (Use["SVM_Lin"])     histSVML   = new TH1F( "MVA_SVM_Lin", "MVA_SVM_Lin",
    nbin,  0.0, 1.0 );
if (Use["FDA_MT"])      histFDAMT  = new TH1F( "MVA_FDA_MT", "MVA_FDA_MT",
    nbin, -2.0, 3.0 );
if (Use["FDA_GA"])      histFDAGA  = new TH1F( "MVA_FDA_GA", "MVA_FDA_GA",
    nbin, -2.0, 3.0 );
```

```
if (Use["Category"])      histCat     = new TH1F( "MVA_Category", "MVA_Category",
    nbin, -2., 2. );
if (Use["Plugin"])        histPBdt    = new TH1F( "MVA_PBDT",     "MVA_BDT",
    nbin, -0.8, 0.8 );

// PDEFoam also returns per-event error, fill in histogram, and also fill significance
if (Use["PDEFoam"]) {
histPDEFoam  = new TH1F( "MVA_PDEFoam", "MVA_PDEFoam",          nbin,  0, 1 );
histPDEFoamErr = new TH1F( "MVA_PDEFoamErr", "MVA_PDEFoam error", nbin, 0, 1 );
histPDEFoamSig = new TH1F( "MVA_PDEFoamSig", "MVA_PDEFoam significance", nbin, 0,
    10 );
}

// Book example histogram for probability (the other methods are done similarly)
TH1F *probHistFi(0), *rarityHistFi(0);
if (Use["Fisher"]) {
probHistFi   = new TH1F( "MVA_Fisher_Proba", "MVA_Fisher_Proba", nbin, 0, 1 );
rarityHistFi = new TH1F( "MVA_Fisher_Rarity", "MVA_Fisher_Rarity", nbin, 0, 1 );
}

// Prepare input tree (this must be replaced by your data source)
// in this example, there is a toy tree with signal and one with background events
// we'll later on use only the "signal" events for the test in this example.
//
TFile *input(0);
TString fname = "./tmva_kstar_bkgunlike.root";
if (!gSystem->AccessPathName( fname ))
input = TFile::Open( fname ); // check if file in local directory exists
else
input = TFile::Open( "http://root.cern.ch/files/tmva_class_example.root" ); // if not:
    download from ROOT server

if (!input) {
std::cout << "ERROR: could not open data file" << std::endl;
exit(1);
}
std::cout << "--- TMVAClassificationApp : Using input file: " << input->GetName() <<
    std::endl;

// Event loop

// Prepare the event tree
// - Here the variable names have to corresponds to your tree
// - You can use the same variables as above which is slightly faster,
//   but of course you can use different ones and copy the values inside the event loop
//
std::cout << "--- Select signal sample" << std::endl;
//TTree* theTree = (TTree*)input->Get("tree_unlike");
TTree* theTree = (TTree*)input->Get("tree_bkg");
/* Float_t userVar1, userVar2;
theTree->SetBranchAddress( "var1", &userVar1 );
```

```
theTree->SetBranchAddress( "var2", &userVar2 );
theTree->SetBranchAddress( "var3", &var3 );
theTree->SetBranchAddress( "var4", &var4 );*/

theTree->SetBranchAddress("invmass",&invmass_u);
theTree->SetBranchAddress("daughter1pt",&d1pt_u);
theTree->SetBranchAddress("daughter2pt",&d2pt_u);
theTree->SetBranchAddress("motherpt",&motherpt_u);
theTree->SetBranchAddress("etad1",&etad1_u);
theTree->SetBranchAddress("etad2",&etad2_u);
theTree->SetBranchAddress("etadm",&etadm_u);
theTree->SetBranchAddress("costs",&costs_u);
theTree->SetBranchAddress("costs1",&costs1_u);
theTree->SetBranchAddress("dcad1",&dcad1_u);
theTree->SetBranchAddress("dcad2",&dcad2_u);
```

```
// Efficiency calculator for cut method
Int_t      nSelCutsGA = 0;
Double_t effS         = 0.7;

std::vector<Float_t> vecVar(4); // vector for EvaluateMVA tests

std::cout << "--- Processing: " << theTree->GetEntries() << " events" << std::endl;
TStopwatch sw;
sw.Start();
for (Long64_t ievt=0; ievt<theTree->GetEntries();ievt++) {

if (ievt%1000 == 0) std::cout << "--- ... Processing event: " << ievt << std::endl;

theTree->GetEntry(ievt);
/*
var1 = userVar1 + userVar2;
var2 = userVar1 - userVar2;
*/
// Return the MVA outputs and fill into histograms

if (Use["CutsGA"]) {
// Cuts is a special case: give the desired signal efficienciy
Bool_t passed = reader->EvaluateMVA( "CutsGA method", effS );
if (passed) nSelCutsGA++;
}

if (Use["Likelihood"    ])   histLk     ->Fill( reader->EvaluateMVA( "Likelihood method" )
    );
if (Use["LikelihoodD" ])   histLkD    ->Fill( reader->EvaluateMVA( "LikelihoodD method"
    ) );
```

```
if (Use["LikelihoodPCA"]) histLkPCA ->Fill( reader->EvaluateMVA( "LikelihoodPCA
    method" ) );
if (Use["LikelihoodKDE"]) histLkKDE ->Fill( reader->EvaluateMVA( "LikelihoodKDE
    method" ) );
if (Use["LikelihoodMIX"]) histLkMIX ->Fill( reader->EvaluateMVA( "LikelihoodMIX
    method" ) );
if (Use["PDERS"       ]) histPD    ->Fill( reader->EvaluateMVA( "PDERS method"  ) );
if (Use["PDERSD"      ]) histPDD   ->Fill( reader->EvaluateMVA( "PDERSD method" ) );
if (Use["PDERSPCA"    ]) histPDPCA ->Fill( reader->EvaluateMVA( "PDERSPCA
    method" ) );
if (Use["KNN"         ]) histKNN   ->Fill( reader->EvaluateMVA( "KNN method"    ) );
if (Use["HMatrix"     ]) histHm    ->Fill( reader->EvaluateMVA( "HMatrix method" ) );
if (Use["Fisher"      ]) histFi    ->Fill( reader->EvaluateMVA( "Fisher method"  ) );
if (Use["FisherG"     ]) histFiG   ->Fill( reader->EvaluateMVA( "FisherG method" ) );
if (Use["BoostedFisher"]) histFiB  ->Fill( reader->EvaluateMVA( "BoostedFisher
    method" ) );
if (Use["LD"          ]) histLD    ->Fill( reader->EvaluateMVA( "LD method"      ) );
if (Use["MLP"         ]) histNn    ->Fill( reader->EvaluateMVA( "MLP method"     ) );
if (Use["MLPBFGS"     ]) histNnbfgs ->Fill( reader->EvaluateMVA( "MLPBFGS method" )
    );
if (Use["MLPBNN"      ]) histNnbnn ->Fill( reader->EvaluateMVA( "MLPBNN method" ) );
if (Use["CFMlpANN"    ]) histNnC   ->Fill( reader->EvaluateMVA( "CFMlpANN method"
    ) );
if (Use["TMlpANN"     ]) histNnT   ->Fill( reader->EvaluateMVA( "TMlpANN method" )
    );
if (Use["DNN"         ]) histNdn   ->Fill( reader->EvaluateMVA( "DNN method"    ) );

if (Use["BDTG"        ]) histBdtG  ->Fill( reader->EvaluateMVA( "BDTG method"   ) );
if (Use["BDTB"        ]) histBdtB  ->Fill( reader->EvaluateMVA( "BDTB method"   ) );
if (Use["BDTD"        ]) histBdtD  ->Fill( reader->EvaluateMVA( "BDTD method"   ) );
if (Use["BDTF"        ]) histBdtF  ->Fill( reader->EvaluateMVA( "BDTF method"   ) );
if (Use["RuleFit"     ]) histRf    ->Fill( reader->EvaluateMVA( "RuleFit method" ) );
if (Use["SVM_Gauss"   ]) histSVMG  ->Fill( reader->EvaluateMVA( "SVM_Gauss method" )
    );
if (Use["SVM_Poly"    ]) histSVMP  ->Fill( reader->EvaluateMVA( "SVM_Poly method" ) );
if (Use["SVM_Lin"     ]) histSVML  ->Fill( reader->EvaluateMVA( "SVM_Lin method" ) );
if (Use["FDA_MT"      ]) histFDAMT ->Fill( reader->EvaluateMVA( "FDA_MT method" )
    );
if (Use["FDA_GA"      ]) histFDAGA ->Fill( reader->EvaluateMVA( "FDA_GA method" )
    );
if (Use["Category"    ]) histCat   ->Fill( reader->EvaluateMVA( "Category method" ) );
if (Use["Plugin"      ]) histPBdt  ->Fill( reader->EvaluateMVA( "P_BDT method"   ) );

if (Use["BDT"         ])
{
histBdt     ->Fill( reader->EvaluateMVA( "BDT method"     ) );
if(reader->EvaluateMVA( "BDT method"      )>cutvalue)
{

hinvmass_signal->Fill(invmass_u);
```

```
}
else
{
hinvmass_bkg->Fill(invmass_u);
}

}
// Retrieve also per-event error
if (Use["PDEFoam"]) {
Double_t val = reader->EvaluateMVA( "PDEFoam method" );
Double_t err = reader->GetMVAError();
histPDEFoam ->Fill( val );
histPDEFoamErr->Fill( err );
if (err>1.e-50) histPDEFoamSig->Fill( val/err );
}

// Retrieve probability instead of MVA output
if (Use["Fisher"])    {
probHistFi  ->Fill( reader->GetProba ( "Fisher method" ) );
rarityHistFi->Fill( reader->GetRarity( "Fisher method" ) );
}
}

// Get elapsed time
sw.Stop();
std::cout << "--- End of event loop: "; sw.Print();

hinvmass_signal->Draw();
//hinvmass_bkg->Draw("same");

// Get efficiency for cuts classifier
if (Use["CutsGA"]) std::cout << "--- Efficiency for CutsGA method: " <<
    double(nSelCutsGA)/theTree->GetEntries()
<< " (for a required signal efficiency of " << effS << ")" << std::endl;

if (Use["CutsGA"]) {

// test: retrieve cuts for particular signal efficiency
// CINT ignores dynamic_casts so we have to use a cuts-secific Reader function to acces the
    pointer
TMVA::MethodCuts* mcuts = reader->FindCutsMVA( "CutsGA method" ) ;

if (mcuts) {
std::vector<Double_t> cutsMin;
std::vector<Double_t> cutsMax;
mcuts->GetCuts( 0.7, cutsMin, cutsMax );
std::cout << "--- ------------------------------" << std::endl;
std::cout << "--- Retrieve cut values for signal efficiency of 0.7 from Reader" << std::endl;
for (UInt_t ivar=0; ivar<cutsMin.size(); ivar++) {
std::cout << "... Cut: "
<< cutsMin[ivar]
```

```
<< " < \""
<< mcuts->GetInputVar(ivar)
<< "\" <= "
<< cutsMax[ivar] << std::endl;
}
std::cout << "--- ------------------------" << std::endl;
}
}

// Write histograms
cout<<"THE CUT VALUE APPLIED
     IS============================"<<cutvalue<<endl;
// TFile *target  = new TFile( "TMVApp_sig.root","RECREATE" );
TFile *target  = new TFile( "TMVApp_bkg.root","RECREATE" );
if (Use["Likelihood"    ])    histLk      ->Write();
if (Use["LikelihoodD"   ])    histLkD     ->Write();
if (Use["LikelihoodPCA"])  histLkPCA  ->Write();
if (Use["LikelihoodKDE"])  histLkKDE  ->Write();
if (Use["LikelihoodMIX"])  histLkMIX  ->Write();
if (Use["PDERS"         ])    histPD      ->Write();
if (Use["PDERSD"        ])    histPDD     ->Write();
if (Use["PDERSPCA"      ])    histPDPCA   ->Write();
if (Use["KNN"           ])    histKNN     ->Write();
if (Use["HMatrix"       ])    histHm      ->Write();
if (Use["Fisher"        ])    histFi      ->Write();
if (Use["FisherG"       ])    histFiG     ->Write();
if (Use["BoostedFisher"])  histFiB     ->Write();
if (Use["LD"            ])    histLD      ->Write();
if (Use["MLP"           ])    histNn      ->Write();
if (Use["MLPBFGS"       ])    histNnbfgs  ->Write();
if (Use["MLPBNN"        ])    histNnbnn   ->Write();
if (Use["CFMlpANN"      ])    histNnC     ->Write();
if (Use["TMlpANN"       ])    histNnT     ->Write();
if (Use["DNN"           ])    histNdn     ->Write();
if (Use["BDT"           ]) {
histBdt     ->Write();hinvmass_signal->Write();hinvmass_bkg->Write();}
if (Use["BDTG"          ])    histBdtG    ->Write();
if (Use["BDTB"          ])    histBdtB    ->Write();
if (Use["BDTD"          ])    histBdtD    ->Write();
if (Use["BDTF"          ])    histBdtF    ->Write();
if (Use["RuleFit"       ])    histRf      ->Write();
if (Use["SVM_Gauss"     ])    histSVMG    ->Write();
if (Use["SVM_Poly"      ])    histSVMP    ->Write();
if (Use["SVM_Lin"       ])    histSVML    ->Write();
if (Use["FDA_MT"        ])    histFDAMT   ->Write();
if (Use["FDA_GA"        ])    histFDAGA   ->Write();
if (Use["Category"      ])    histCat     ->Write();
if (Use["Plugin"        ])    histPBdt    ->Write();

// Write also error and significance histos
```

```
if (Use["PDEFoam"]) { histPDEFoam->Write(); histPDEFoamErr->Write();
    histPDEFoamSig->Write(); }

// Write also probability hists
if (Use["Fisher"]) { if (probHistFi != 0) probHistFi->Write(); if (rarityHistFi != 0)
    rarityHistFi->Write(); }
target->Close();

std::cout << "--- Created root file: \"TMVApp.root\" containing the MVA output
    histograms" << std::endl;

delete reader;

std::cout << "==> TMVAClassificationApplication is done!" << std::endl << std::endl;
}

int main( int argc, char** argv )
{
TString methodList;
for (int i=1; i<argc; i++) {
TString regMethod(argv[i]);
if(regMethod=="-b" || regMethod=="--batch") continue;
if (!methodList.IsNull()) methodList += TString(",");
methodList += regMethod;
}
TMVAClassificationApplication_kstar(methodList);
return 0;
}
```

8.3.2 TMVA application code for K^{*0}.

```
/// \file
/// \ingroup tutorial_tmva
/// \notebook -nodraw
/// This macro provides a simple example on how to use the trained classifiers
/// within an analysis module
/// - Project   : TMVA - a Root-integrated toolkit for multivariate data analysis
/// - Package   : TMVA
/// - Exectuable: TMVAClassificationApplication
///
/// \macro_output
/// \macro_code
/// \author Andreas Hoecker

/*
root -l ./TMVAClassificationApplication_kstar.C\(\"BDT\"\)
*/
#include <cstdlib>
```

```
#include <vector>
#include <iostream>
#include <map>
#include <string>

#include "TFile.h"
#include "TTree.h"
#include "TString.h"
#include "TSystem.h"
#include "TROOT.h"
#include "TStopwatch.h"

#include "TMVA/Tools.h"
#include "TMVA/Reader.h"
#include "TMVA/MethodCuts.h"

using namespace TMVA;

void TMVAClassificationApplication_kstar( TString myMethodList = "" )
{

double cutvalue=-0.0932; //like
cout<<"THE CUT VALUE APPLIED
    IS========================"<<cutvalue<<endl;


//---------------------------------
// This loads the library
TMVA::Tools::Instance();

// Default MVA methods to be trained + tested
std::map<std::string,int> Use;

// Cut optimisation
Use["Cuts"]            = 1;
Use["CutsD"]           = 1;
Use["CutsPCA"]         = 0;
Use["CutsGA"]          = 0;
Use["CutsSA"]          = 0;
//
// 1-dimensional likelihood ("naive Bayes estimator")
Use["Likelihood"]      = 1;
Use["LikelihoodD"]     = 0; // the "D" extension indicates decorrelated input variables (see
    option strings)
Use["LikelihoodPCA"]   = 1; // the "PCA" extension indicates PCA-transformed input variables
    (see option strings)
Use["LikelihoodKDE"]   = 0;
Use["LikelihoodMIX"]   = 0;
//
// Mutidimensional likelihood and Nearest-Neighbour methods
Use["PDERS"]           = 1;
```

```
Use["PDERSD"]        = 0;
Use["PDERSPCA"]      = 0;
Use["PDEFoam"]       = 1;
Use["PDEFoamBoost"]  = 0; // uses generalised MVA method boosting
Use["KNN"]           = 1; // k-nearest neighbour method
//
// Linear Discriminant Analysis
Use["LD"]            = 1; // Linear Discriminant identical to Fisher
Use["Fisher"]        = 0;
Use["FisherG"]       = 0;
Use["BoostedFisher"] = 0; // uses generalised MVA method boosting
Use["HMatrix"]       = 0;
//
// Function Discriminant analysis
Use["FDA_GA"]        = 1; // minimisation of user-defined function using Genetics Algorithm
Use["FDA_SA"]        = 0;
Use["FDA_MC"]        = 0;
Use["FDA_MT"]        = 0;
Use["FDA_GAMT"]      = 0;
Use["FDA_MCMT"]      = 0;
//
// Neural Networks (all are feed-forward Multilayer Perceptrons)
Use["MLP"]           = 0; // Recommended ANN
Use["MLPBFGS"]       = 0; // Recommended ANN with optional training method
Use["MLPBNN"]        = 1; // Recommended ANN with BFGS training method and bayesian
    regulator
Use["CFMlpANN"]      = 0; // Depreciated ANN from ALEPH
Use["TMlpANN"]       = 0; // ROOT's own ANN
Use["DNN"]           = 0; // improved implementation of a NN
//
// Support Vector Machine
Use["SVM"]           = 1;
//
// Boosted Decision Trees
Use["BDT"]           = 1; // uses Adaptive Boost
Use["BDTG"]          = 0; // uses Gradient Boost
Use["BDTB"]          = 0; // uses Bagging
Use["BDTD"]          = 0; // decorrelation + Adaptive Boost
Use["BDTF"]          = 0; // allow usage of fisher discriminant for node splitting
//
// Friedman's RuleFit method, ie, an optimised series of cuts ("rules")
Use["RuleFit"]       = 1;
// ---------------------------------------------------------------
Use["Plugin"]        = 0;
Use["Category"]      = 0;
Use["SVM_Gauss"]     = 0;
Use["SVM_Poly"]      = 0;
Use["SVM_Lin"]       = 0;

std::cout << std::endl;
std::cout << "==> Start TMVAClassificationApplication" << std::endl;
```

```
// Select methods (don't look at this code - not of interest)
if (myMethodList != "") {
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) it->second
    = 0;

std::vector<TString> mlist = gTools().SplitString( myMethodList, ',' );
for (UInt_t i=0; i<mlist.size(); i++) {
std::string regMethod(mlist[i]);

if (Use.find(regMethod) == Use.end()) {
std::cout << "Method \"" << regMethod
<< "\" not known in TMVA under this name. Choose among the following:" << std::endl;
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) {
std::cout << it->first << " ";
}
std::cout << std::endl;
return;
}
Use[regMethod] = 1;
}
}

//---------------------------------

// Create the Reader object

TMVA::Reader *reader = new TMVA::Reader( "!Color:!Silent" );

// Create a set of variables and declare them to the reader
// - the variable names MUST corresponds in name and type to those given in the weight file(s)
    used
/*
Float_t var1, var2;
Float_t var3, var4;
reader->AddVariable( "myvar1 := var1+var2", &var1 );
reader->AddVariable( "myvar2 := var1-var2", &var2 );
reader->AddVariable( "var3",                &var3 );
reader->AddVariable( "var4",                &var4 );
*/


float
    invmass_u,d1pt_u,d2pt_u,motherpt_u,etad1_u,etad2_u,etadm_u,costs_u,costs1_u,dcad1_u,dcad2_u;
// reader->AddVariable("invmass",&invmass_u);
// reader->AddVariable("daughter1pt",&d1pt_u);
//reader->AddVariable("daughter2pt",&d2pt_u);
reader->AddVariable("motherpt",&motherpt_u);
//reader->AddVariable("etad1",&etad1_u);
//reader->AddVariable("etad2",&etad2_u);
reader->AddVariable("etadm",&etadm_u);
```

```
reader->AddVariable("costs",&costs_u);
//reader->AddVariable("costs1",&costs1_u);
reader->AddVariable("dcad1",&dcad1_u);
//reader->AddVariable("dcad2",&dcad2_u);

// Spectator variables declared in the training have to be added to the reader, too
/*
Float_t spec1,spec2;
reader->AddSpectator( "spec1 := var1*2", &spec1 );
reader->AddSpectator( "spec2 := var1*3", &spec2 );
*/
/* Float_t Category_cat1, Category_cat2, Category_cat3;
if (Use["Category"]){
// Add artificial spectators for distinguishing categories
reader->AddSpectator( "Category_cat1 := var3<=0",          &Category_cat1 );
reader->AddSpectator( "Category_cat2 := (var3>0)&&(var4<0)", &Category_cat2 );
reader->AddSpectator( "Category_cat3 := (var3>0)&&(var4>=0)", &Category_cat3 );
}*/

// Book the MVA methods

TString dir    = "dataset/weights/";
TString prefix = "TMVAClassification";

// Book method(s)
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) {
if (it->second) {
TString methodName = TString(it->first) + TString(" method");
TString weightfile = dir + prefix + TString("_") + TString(it->first) +
    TString(".weights.xml");
reader->BookMVA( methodName, weightfile );
}
}

// Book output histograms
UInt_t nbin = 100;
TH1F *histLk(0), *histLkD(0), *histLkPCA(0), *histLkKDE(0), *histLkMIX(0), *histPD(0),
    *histPDD(0);
TH1F *histPDPCA(0), *histPDEFoam(0), *histPDEFoamErr(0), *histPDEFoamSig(0),
    *histKNN(0), *histHm(0);
TH1F *histFi(0), *histFiG(0), *histFiB(0), *histLD(0),
    *histNn(0),*histNnbfgs(0),*histNnbnn(0);
TH1F *histNnC(0), *histNnT(0), *histNdn(0), *histBdt(0), *histBdtG(0), *histBdtB(0),
    *histBdtD(0);
TH1F *histBdtF(0), *histRf(0), *histSVMG(0), *histSVMP(0), *histSVML(0),
    *histFDAMT(0), *histFDAGA(0);
TH1F *histCat(0), *histPBdt(0);

TH1D *hinvmass_signal=new TH1D("invsig","invsig",60,0.6,1.2);
TH1D *hinvmass_bkg=new TH1D("invbkg","invbkg",60,0.6,1.2);
```

```
if (Use["Likelihood"])     histLk      = new TH1F( "MVA_Likelihood", "MVA_Likelihood",
    nbin, -1, 1 );
if (Use["LikelihoodD"])    histLkD     = new TH1F( "MVA_LikelihoodD", "MVA_LikelihoodD",
    nbin, -1, 0.9999 );
if (Use["LikelihoodPCA"]) histLkPCA = new TH1F( "MVA_LikelihoodPCA",
    "MVA_LikelihoodPCA", nbin, -1, 1 );
if (Use["LikelihoodKDE"]) histLkKDE = new TH1F( "MVA_LikelihoodKDE",
    "MVA_LikelihoodKDE", nbin, -0.00001, 0.99999 );
if (Use["LikelihoodMIX"]) histLkMIX = new TH1F( "MVA_LikelihoodMIX",
    "MVA_LikelihoodMIX", nbin, 0, 1 );
if (Use["PDERS"])          histPD      = new TH1F( "MVA_PDERS",   "MVA_PDERS",
    nbin,   0,  1 );
if (Use["PDERSD"])         histPDD     = new TH1F( "MVA_PDERSD", "MVA_PDERSD",
    nbin,   0,  1 );
if (Use["PDERSPCA"])       histPDPCA = new TH1F( "MVA_PDERSPCA",
    "MVA_PDERSPCA", nbin, 0, 1 );
if (Use["KNN"])            histKNN     = new TH1F( "MVA_KNN",     "MVA_KNN",
    nbin,   0,  1 );
if (Use["HMatrix"])        histHm      = new TH1F( "MVA_HMatrix", "MVA_HMatrix",
    nbin, -0.95, 1.55  );
if (Use["Fisher"])         histFi      = new TH1F( "MVA_Fisher",  "MVA_Fisher",
    nbin, -4, 4 );
if (Use["FisherG"])        histFiG     = new TH1F( "MVA_FisherG", "MVA_FisherG",
    nbin, -1, 1 );
if (Use["BoostedFisher"]) histFiB     = new TH1F( "MVA_BoostedFisher",
    "MVA_BoostedFisher", nbin, -2, 2 );
if (Use["LD"])             histLD      = new TH1F( "MVA_LD",      "MVA_LD",
    nbin, -2, 2 );
if (Use["MLP"])            histNn      = new TH1F( "MVA_MLP",     "MVA_MLP",
    nbin, -1.25, 1.5  );
if (Use["MLPBFGS"])        histNnbfgs  = new TH1F( "MVA_MLPBFGS", "MVA_MLPBFGS",
    nbin, -1.25, 1.5 );
if (Use["MLPBNN"])         histNnbnn   = new TH1F( "MVA_MLPBNN", "MVA_MLPBNN",
    nbin, -1.25, 1.5  );
if (Use["CFMlpANN"])       histNnC     = new TH1F( "MVA_CFMlpANN", "MVA_CFMlpANN",
    nbin, 0, 1 );
if (Use["TMlpANN"])        histNnT     = new TH1F( "MVA_TMlpANN", "MVA_TMlpANN",
    nbin, -1.3, 1.3  );
if (Use["DNN"])            histNdn     = new TH1F( "MVA_DNN",     "MVA_DNN",
    nbin, -0.1, 1.1  );
if (Use["BDT"])            histBdt     = new TH1F( "MVA_BDT",     "MVA_BDT",
    nbin, -0.8, 0.8  );
if (Use["BDTG"])           histBdtG    = new TH1F( "MVA_BDTG",    "MVA_BDTG",
    nbin, -1.0, 1.0  );
if (Use["BDTB"])           histBdtB    = new TH1F( "MVA_BDTB",    "MVA_BDTB",
    nbin, -1.0, 1.0  );
if (Use["BDTD"])           histBdtD    = new TH1F( "MVA_BDTD",    "MVA_BDTD",
    nbin, -0.8, 0.8  );
if (Use["BDTF"])           histBdtF    = new TH1F( "MVA_BDTF",    "MVA_BDTF",
    nbin, -1.0, 1.0  );
```

```
if (Use["RuleFit"])        histRf      = new TH1F( "MVA_RuleFit",   "MVA_RuleFit",
    nbin, -2.0, 2.0 );
if (Use["SVM_Gauss"])      histSVMG    = new TH1F( "MVA_SVM_Gauss", "MVA_SVM_Gauss",
    nbin, 0.0, 1.0 );
if (Use["SVM_Poly"])       histSVMP    = new TH1F( "MVA_SVM_Poly", "MVA_SVM_Poly",
    nbin,  0.0, 1.0 );
if (Use["SVM_Lin"])        histSVML    = new TH1F( "MVA_SVM_Lin", "MVA_SVM_Lin",
    nbin,  0.0, 1.0 );
if (Use["FDA_MT"])         histFDAMT = new TH1F( "MVA_FDA_MT", "MVA_FDA_MT",
    nbin, -2.0, 3.0 );
if (Use["FDA_GA"])         histFDAGA = new TH1F( "MVA_FDA_GA", "MVA_FDA_GA",
    nbin, -2.0, 3.0 );
if (Use["Category"])       histCat     = new TH1F( "MVA_Category", "MVA_Category",
    nbin, -2., 2. );
if (Use["Plugin"])         histPBdt    = new TH1F( "MVA_PBDT",    "MVA_BDT",
    nbin, -0.8, 0.8 );

// PDEFoam also returns per-event error, fill in histogram, and also fill significance
if (Use["PDEFoam"]) {
histPDEFoam  = new TH1F( "MVA_PDEFoam", "MVA_PDEFoam",          nbin,  0, 1 );
histPDEFoamErr = new TH1F( "MVA_PDEFoamErr", "MVA_PDEFoam error", nbin, 0, 1 );
histPDEFoamSig = new TH1F( "MVA_PDEFoamSig", "MVA_PDEFoam significance", nbin, 0,
    10 );
}

// Book example histogram for probability (the other methods are done similarly)
TH1F *probHistFi(0), *rarityHistFi(0);
if (Use["Fisher"]) {
probHistFi   = new TH1F( "MVA_Fisher_Proba", "MVA_Fisher_Proba", nbin, 0, 1 );
rarityHistFi = new TH1F( "MVA_Fisher_Rarity", "MVA_Fisher_Rarity", nbin, 0, 1 );
}

// Prepare input tree (this must be replaced by your data source)
// in this example, there is a toy tree with signal and one with background events
// we'll later on use only the "signal" events for the test in this example.
//
TFile *input(0);
TString fname = "./tmva_kstar_bkgunlike.root";
if (!gSystem->AccessPathName( fname ))
input = TFile::Open( fname ); // check if file in local directory exists
else
input = TFile::Open( "http://root.cern.ch/files/tmva_class_example.root" ); // if not:
    download from ROOT server

if (!input) {
std::cout << "ERROR: could not open data file" << std::endl;
exit(1);
}
std::cout << "--- TMVAClassificationApp : Using input file: " << input->GetName() <<
    std::endl;
```

```
// Event loop

// Prepare the event tree
// - Here the variable names have to corresponds to your tree
// - You can use the same variables as above which is slightly faster,
//   but of course you can use different ones and copy the values inside the event loop
//
std::cout << "--- Select signal sample" << std::endl;
//TTree* theTree = (TTree*)input->Get("tree_unlike");
TTree* theTree = (TTree*)input->Get("tree_bkg");
/* Float_t userVar1, userVar2;
theTree->SetBranchAddress( "var1", &userVar1 );
theTree->SetBranchAddress( "var2", &userVar2 );
theTree->SetBranchAddress( "var3", &var3 );
theTree->SetBranchAddress( "var4", &var4 );*/

theTree->SetBranchAddress("invmass",&invmass_u);
theTree->SetBranchAddress("daughter1pt",&d1pt_u);
theTree->SetBranchAddress("daughter2pt",&d2pt_u);
theTree->SetBranchAddress("motherpt",&motherpt_u);
theTree->SetBranchAddress("etad1",&etad1_u);
theTree->SetBranchAddress("etad2",&etad2_u);
theTree->SetBranchAddress("etadm",&etadm_u);
theTree->SetBranchAddress("costs",&costs_u);
theTree->SetBranchAddress("costs1",&costs1_u);
theTree->SetBranchAddress("dcad1",&dcad1_u);
theTree->SetBranchAddress("dcad2",&dcad2_u);




// Efficiency calculator for cut method
Int_t     nSelCutsGA = 0;
Double_t effS        = 0.7;

std::vector<Float_t> vecVar(4); // vector for EvaluateMVA tests

std::cout << "--- Processing: " << theTree->GetEntries() << " events" << std::endl;
TStopwatch sw;
sw.Start();
for (Long64_t ievt=0; ievt<theTree->GetEntries();ievt++) {

if (ievt%1000 == 0) std::cout << "--- ... Processing event: " << ievt << std::endl;

theTree->GetEntry(ievt);
/*
var1 = userVar1 + userVar2;
```

```
var2 = userVar1 - userVar2;
*/
// Return the MVA outputs and fill into histograms

if (Use["CutsGA"]) {
// Cuts is a special case: give the desired signal efficiency
Bool_t passed = reader->EvaluateMVA( "CutsGA method", effS );
if (passed) nSelCutsGA++;
}

if (Use["Likelihood"   ]) histLk      ->Fill( reader->EvaluateMVA( "Likelihood method" ) );
if (Use["LikelihoodD"  ]) histLkD     ->Fill( reader->EvaluateMVA( "LikelihoodD method" ) );
if (Use["LikelihoodPCA"]) histLkPCA   ->Fill( reader->EvaluateMVA( "LikelihoodPCA method" ) );
if (Use["LikelihoodKDE"]) histLkKDE   ->Fill( reader->EvaluateMVA( "LikelihoodKDE method" ) );
if (Use["LikelihoodMIX"]) histLkMIX   ->Fill( reader->EvaluateMVA( "LikelihoodMIX method" ) );
if (Use["PDERS"        ]) histPD      ->Fill( reader->EvaluateMVA( "PDERS method" ) );
if (Use["PDERSD"       ]) histPDD     ->Fill( reader->EvaluateMVA( "PDERSD method" ) );
if (Use["PDERSPCA"     ]) histPDPCA   ->Fill( reader->EvaluateMVA( "PDERSPCA method" ) );
if (Use["KNN"          ]) histKNN     ->Fill( reader->EvaluateMVA( "KNN method"     ) );
if (Use["HMatrix"      ]) histHm      ->Fill( reader->EvaluateMVA( "HMatrix method" ) );
if (Use["Fisher"       ]) histFi      ->Fill( reader->EvaluateMVA( "Fisher method"  ) );
if (Use["FisherG"      ]) histFiG     ->Fill( reader->EvaluateMVA( "FisherG method" ) );
if (Use["BoostedFisher"]) histFiB     ->Fill( reader->EvaluateMVA( "BoostedFisher method" ) );
if (Use["LD"           ]) histLD      ->Fill( reader->EvaluateMVA( "LD method"      ) );
if (Use["MLP"          ]) histNn      ->Fill( reader->EvaluateMVA( "MLP method"     ) );
if (Use["MLPBFGS"      ]) histNnbfgs  ->Fill( reader->EvaluateMVA( "MLPBFGS method" ) );
if (Use["MLPBNN"       ]) histNnbnn   ->Fill( reader->EvaluateMVA( "MLPBNN method" ) );
if (Use["CFMlpANN"     ]) histNnC     ->Fill( reader->EvaluateMVA( "CFMlpANN method" ) );
if (Use["TMlpANN"      ]) histNnT     ->Fill( reader->EvaluateMVA( "TMlpANN method" ) );
if (Use["DNN"          ]) histNdn     ->Fill( reader->EvaluateMVA( "DNN method"     ) );

if (Use["BDTG"         ]) histBdtG    ->Fill( reader->EvaluateMVA( "BDTG method"    ) );
if (Use["BDTB"         ]) histBdtB    ->Fill( reader->EvaluateMVA( "BDTB method"    ) );
if (Use["BDTD"         ]) histBdtD    ->Fill( reader->EvaluateMVA( "BDTD method"    ) );
if (Use["BDTF"         ]) histBdtF    ->Fill( reader->EvaluateMVA( "BDTF method"    ) );
if (Use["RuleFit"      ]) histRf      ->Fill( reader->EvaluateMVA( "RuleFit method" ) );
if (Use["SVM_Gauss"    ]) histSVMG    ->Fill( reader->EvaluateMVA( "SVM_Gauss method" ) );
if (Use["SVM_Poly"     ]) histSVMP    ->Fill( reader->EvaluateMVA( "SVM_Poly method" ) );
if (Use["SVM_Lin"      ]) histSVML    ->Fill( reader->EvaluateMVA( "SVM_Lin method" ) );
```

```
if (Use["FDA_MT"]) histFDAMT ->Fill( reader->EvaluateMVA( "FDA_MT method" )
    );
if (Use["FDA_GA"]) histFDAGA ->Fill( reader->EvaluateMVA( "FDA_GA method" )
    );
if (Use["Category"]) histCat ->Fill( reader->EvaluateMVA( "Category method" ) );
if (Use["Plugin"]) histPBdt ->Fill( reader->EvaluateMVA( "P_BDT method" ) );

if (Use["BDT"])
{
histBdt ->Fill( reader->EvaluateMVA( "BDT method" ) );
if(reader->EvaluateMVA( "BDT method" )>cutvalue)
{

hinvmass_signal->Fill(invmass_u);

}
else
{
hinvmass_bkg->Fill(invmass_u);
}

}
// Retrieve also per-event error
if (Use["PDEFoam"]) {
Double_t val = reader->EvaluateMVA( "PDEFoam method" );
Double_t err = reader->GetMVAError();
histPDEFoam ->Fill( val );
histPDEFoamErr->Fill( err );
if (err>1.e-50) histPDEFoamSig->Fill( val/err );
}

// Retrieve probability instead of MVA output
if (Use["Fisher"]) {
probHistFi ->Fill( reader->GetProba ( "Fisher method" ) );
rarityHistFi->Fill( reader->GetRarity( "Fisher method" ) );
}
}

// Get elapsed time
sw.Stop();
std::cout << "--- End of event loop: "; sw.Print();

hinvmass_signal->Draw();
//hinvmass_bkg->Draw("same");

// Get efficiency for cuts classifier
if (Use["CutsGA"]) std::cout << "--- Efficiency for CutsGA method: " <<
    double(nSelCutsGA)/theTree->GetEntries()
<< " (for a required signal efficiency of " << effS << ")" << std::endl;

if (Use["CutsGA"]) {
```

```
// test: retrieve cuts for particular signal efficiency
// CINT ignores dynamic_casts so we have to use a cuts-secific Reader function to acces the
    pointer
TMVA::MethodCuts* mcuts = reader->FindCutsMVA( "CutsGA method" ) ;


if (mcuts) {
std::vector<Double_t> cutsMin;
std::vector<Double_t> cutsMax;
mcuts->GetCuts( 0.7, cutsMin, cutsMax );
std::cout << "--- ----------------------------" << std::endl;
std::cout << "--- Retrieve cut values for signal efficiency of 0.7 from Reader" << std::endl;
for (UInt_t ivar=0; ivar<cutsMin.size(); ivar++) {
std::cout << "... Cut: "
<< cutsMin[ivar]
<< " < \""
<< mcuts->GetInputVar(ivar)
<< "\" <= "
<< cutsMax[ivar] << std::endl;
}
std::cout << "--- ----------------------------" << std::endl;
}
}



// Write histograms
cout<<"THE CUT VALUE APPLIED IS====================="<<cutvalue<<endl;
// TFile *target  = new TFile( "TMVApp_sig.root","RECREATE" );
TFile *target  = new TFile( "TMVApp_bkg.root","RECREATE" );
if (Use["Likelihood"    ])   histLk      ->Write();
if (Use["LikelihoodD"   ])   histLkD     ->Write();
if (Use["LikelihoodPCA"])  histLkPCA  ->Write();
if (Use["LikelihoodKDE"])  histLkKDE  ->Write();
if (Use["LikelihoodMIX"])  histLkMIX  ->Write();
if (Use["PDERS"         ])   histPD      ->Write();
if (Use["PDERSD"        ])   histPDD     ->Write();
if (Use["PDERSPCA"      ])   histPDPCA   ->Write();
if (Use["KNN"           ])   histKNN     ->Write();
if (Use["HMatrix"       ])   histHm      ->Write();
if (Use["Fisher"        ])   histFi      ->Write();
if (Use["FisherG"       ])   histFiG     ->Write();
if (Use["BoostedFisher"])  histFiB     ->Write();
if (Use["LD"            ])   histLD      ->Write();
if (Use["MLP"           ])   histNn      ->Write();
if (Use["MLPBFGS"       ])   histNnbfgs  ->Write();
if (Use["MLPBNN"        ])   histNnbnn   ->Write();
if (Use["CFMlpANN"      ])   histNnC     ->Write();
if (Use["TMlpANN"       ])   histNnT     ->Write();
```

```
if (Use["DNN"          ])     histNdn    ->Write();
if (Use["BDT"          ]){    histBdt    ->Write();
hinvmass_signal->Write();hinvmass_bkg->Write();}
if (Use["BDTG"         ])     histBdtG   ->Write();
if (Use["BDTB"         ])     histBdtB   ->Write();
if (Use["BDTD"         ])     histBdtD   ->Write();
if (Use["BDTF"         ])     histBdtF   ->Write();
if (Use["RuleFit"      ])     histRf     ->Write();
if (Use["SVM_Gauss"    ])     histSVMG   ->Write();
if (Use["SVM_Poly"     ])     histSVMP   ->Write();
if (Use["SVM_Lin"      ])     histSVML   ->Write();
if (Use["FDA_MT"       ])     histFDAMT  ->Write();
if (Use["FDA_GA"       ])     histFDAGA  ->Write();
if (Use["Category"     ])     histCat    ->Write();
if (Use["Plugin"       ])     histPBdt   ->Write();


// Write also error and significance histos
if (Use["PDEFoam"]) { histPDEFoam->Write(); histPDEFoamErr->Write();
     histPDEFoamSig->Write(); }

// Write also probability hists
if (Use["Fisher"]) { if (probHistFi != 0) probHistFi->Write(); if (rarityHistFi != 0)
     rarityHistFi->Write(); }
target->Close();

std::cout << "--- Created root file: \"TMVApp.root\" containing the MVA output
     histograms" << std::endl;

delete reader;

std::cout << "==> TMVAClassificationApplication is done!" << std::endl << std::endl;
}


int main( int argc, char** argv )
{
TString methodList;
for (int i=1; i<argc; i++) {
TString regMethod(argv[i]);
if(regMethod=="-b" || regMethod=="--batch") continue;
if (!methodList.IsNull()) methodList += TString(",");
methodList += regMethod;
}
TMVAClassificationApplication_kstar(methodList);
return 0;
}
```

8.3.3 Program for extracting signal after application for K^{*0}.

```
Double_t BWPoly2(Double_t *x, Double_t *par)
{
return (par[0]*par[1]*0.01*1)/(2*3.14159)/((x[0]-par[2])*(x[0]-par[2])+(par[1]*par[1])/4.)+
     par[3]+ par[4]*x[0]+par[5]*x[0]*x[0];

}

Double_t BW(Double_t *x, Double_t *par)
{
return (par[0]*par[1]*0.01*1)/(2*3.14159)/((x[0]-par[2])*(x[0]-par[2])+(par[1]*par[1])/4.);
}

Double_t Pol2(Double_t *x, Double_t *par)
{
return par[0]+ par[1]*x[0]+par[2]*x[0]*x[0];
}


void invmass()
{
gStyle->SetOptFit(1);
gStyle->SetOptTitle(0);
gStyle->SetOptStat(0);
gStyle->SetStatColor(0);
gStyle->SetCanvasColor(0);
gStyle->SetPadColor(0);
gStyle->SetPadBorderMode(0);
gStyle->SetCanvasBorderMode(0);
gStyle->SetFrameBorderMode(0);
gStyle->SetFillColor(0);
gStyle->SetLineColor(1);
gStyle->SetTitleTextColor(1);
gStyle->SetTitleColor(1);

TCanvas *cinv[2];
for(Int_t ip = 0; ip < 2; ip++)
{
cinv[ip] = new TCanvas(Form("cinv%d",ip),"cinv",10,10,600,600);
cinv[ip]->SetLeftMargin(0.2);
cinv[ip]->SetRightMargin(0.05);
cinv[ip]->SetBottomMargin(0.2);
}

////sig eff=0.93////
TFile *f1 = new TFile("TMVApp_sig.root");
TH1F *hsig =(TH1F*)f1->Get("invsig");
TH1F *hsig1 =(TH1F*)f1->Get("invbkg");
```

```
TH1F *htmva=(TH1F*)hsig->Clone("htmva");
htmva->Sumw2();

TFile *f2 = new TFile("TMVApp_bkg.root");
TH1F *hbkg =(TH1F*)f2->Get("invsig");
TH1F *hbkg1 =(TH1F*)f2->Get("invbkg");
TH1F *htmva_bkg=(TH1F*)hbkg->Clone("htmva_bkg");
htmva_bkg->Sumw2();


htmva->GetXaxis()->SetTitle("M_{k#pi} (GeV/c^{2})");
htmva->GetXaxis()->SetTitleSize(0.05);
htmva->GetXaxis()->SetTitleOffset(1.03);
htmva->GetXaxis()->SetTitleFont(42);
htmva->GetXaxis()->CenterTitle(true);
htmva->GetXaxis()->SetLabelSize(0.05);
htmva->GetXaxis()->SetLabelFont(42);
htmva->GetXaxis()->SetNdivisions(511);
htmva->GetYaxis()->SetTitle("Counts(/10 MeV/c^{2})");
htmva->GetYaxis()->SetTitleFont(42);
htmva->GetYaxis()->CenterTitle(true);
htmva->GetYaxis()->SetTitleSize(0.05);
htmva->GetYaxis()->SetTitleOffset(1.7);
htmva->GetYaxis()->SetLabelSize(0.05);
htmva->GetYaxis()->SetLabelFont(42);
htmva->GetYaxis()->SetNdivisions(509);


htmva_bkg->GetXaxis()->SetTitle("M_{k#pi}");
htmva_bkg->GetXaxis()->SetTitleSize(0.05);
htmva_bkg->GetXaxis()->SetTitleOffset(1.03);
htmva_bkg->GetXaxis()->SetTitleFont(42);
htmva_bkg->GetXaxis()->CenterTitle(true);
htmva_bkg->GetXaxis()->SetLabelSize(0.05);
htmva_bkg->GetXaxis()->SetLabelFont(42);
htmva_bkg->GetXaxis()->SetNdivisions(511);
htmva_bkg->GetYaxis()->SetTitle("Counts(/10 MeV/c^{2})");
htmva_bkg->GetYaxis()->SetTitleFont(42);
htmva_bkg->GetYaxis()->CenterTitle(true);
htmva_bkg->GetYaxis()->SetTitleSize(0.05);
htmva_bkg->GetYaxis()->SetTitleOffset(1.7);
htmva_bkg->GetYaxis()->SetLabelSize(0.05);
htmva_bkg->GetYaxis()->SetLabelFont(42);
htmva_bkg->GetYaxis()->SetNdivisions(509);



htmva->SetMarkerStyle(20);
htmva->SetMarkerColor(1);
htmva->SetLineColor(1);
```

```
htmva_bkg->SetMarkerStyle(20);
htmva_bkg->SetMarkerColor(2);
htmva_bkg->SetLineColor(2);

TH1F *htmva_clone=(TH1F*)htmva->Clone("htmva_clone");
TH1F *htmva_bkg_clone=(TH1F*)htmva_bkg->Clone("htmva_bkg_clone");




float  fitlow=0.69;
float  fithi=1.2;
float  normlow=5;
float  normhigh=10;

htmva_bkg->Scale(htmva->
Integral(normlow,normhigh)/(htmva_bkg->Integral(normlow,normhigh)));
htmva_bkg_clone->Scale(htmva_clone->
Integral(normlow,normhigh)/(htmva_bkg_clone->Integral(normlow,normhigh)));


htmva->Add(htmva_bkg,-1);


TF1 *fitFcn1 = new TF1("fitFcn1",BWPoly2,fitlow,fithi,6);
fitFcn1->SetParameter(0,100);
fitFcn1->FixParameter(1,0.048);
fitFcn1->SetParLimits(2,0.85,0.95);
fitFcn1->SetParameter(3,10.0);
fitFcn1->SetParameter(4,100.0);
fitFcn1->SetParameter(5,10000.0);
fitFcn1->SetLineColor(2);
fitFcn1->SetParNames("Yield","Width","Mass","p0","p1","p2");

htmva->Fit(fitFcn1,"IER+");

float  a=fitFcn1->GetParameter(0);
float  b=fitFcn1->GetParameter(2);
float  c=fitFcn1->GetParameter(3);
float  d=fitFcn1->GetParameter(4);
float  e=fitFcn1->GetParameter(5);


TF1 *fitFcn2 = new TF1("fitFcn2",BW,fitlow,fithi,3);
fitFcn2->FixParameter(0,a);
fitFcn2->FixParameter(1,0.048);
fitFcn2->FixParameter(2,b);
fitFcn2->SetLineColor(209);
fitFcn2->SetLineStyle(2);
```

```
TF1 *fitFcn3 = new TF1("fitFcn3",Pol2,fitlow,fithi,3);
fitFcn3->FixParameter(0,c);
fitFcn3->FixParameter(1,d);
fitFcn3->FixParameter(2,e);
fitFcn3->SetLineColor(4);



cout<<"signal by BW "<<fitFcn2->Integral(0.6,2.0)/0.01<<"\n";
cout<<htmva_clone->Integral(10,50)<<"\n";
cout<<"Significance bdt===="<<(fitFcn2->
Integral (0.6,2.0) /0.01)/sqrt(htmva_clone->Integral(10,50))<<"\n"; //0.7 to 1.1


cinv[0]->cd();
htmva_clone->Draw();
htmva_bkg_clone->Draw("same");
TLegend *l1 = new TLegend(0.2365772,0.7801394,0.3708054,0.8797909);
l1->SetTextFont(42);
l1->SetBorderSize(0);
l1->SetFillStyle(0);
l1->SetFillColor(0);
l1->SetMargin(0.25);
l1->SetTextSize(0.03);
l1->SetEntrySeparation(0.5);
l1->AddEntry(htmva_clone,"Unlike pair after BDT cut","p");
l1->AddEntry(htmva_bkg_clone,"Normalized like pair after BDT cut","p");
l1->Draw();

cinv[1]->cd();
htmva->Draw();
fitFcn3->Draw("same");
fitFcn2->Draw("same");
TLegend *l2 = new TLegend(0.1565772,0.701394,0.508054,0.8797909);
l2->SetTextFont(42);
l2->SetBorderSize(0);
l2->SetFillStyle(0);
l2->SetFillColor(0);
l2->SetMargin(0.25);
l2->SetTextSize(0.03);
l2->SetEntrySeparation(0.5);
l2->AddEntry(fitFcn1,"BW + Pol2","l");
l2->AddEntry(fitFcn2,"BW","l");
l2->AddEntry(fitFcn3,"Pol2","l");
l2->Draw();



}
```

8.3.4 Program for extracting signal by like-sign technique.

```
Double_t BWPoly2(Double_t *x, Double_t *par)
{
return (par[0]*par[1]*0.01*1)/(2*3.14159)/((x[0]-par[2])*(x[0]-par[2])+(par[1]*par[1])/4.)+
     par[3]+ par[4]*x[0]+par[5]*x[0]*x[0];

}

Double_t BW(Double_t *x, Double_t *par)
{
return (par[0]*par[1]*0.01*1)/(2*3.14159)/((x[0]-par[2])*(x[0]-par[2])+(par[1]*par[1])/4.);
}

Double_t Pol2(Double_t *x, Double_t *par)
{
return par[0]+ par[1]*x[0]+par[2]*x[0]*x[0];
}


void invmass_like()
{
gStyle->SetOptFit(1);
gStyle->SetOptTitle(0);
gStyle->SetOptStat(0);
gStyle->SetStatColor(0);
gStyle->SetCanvasColor(0);
gStyle->SetPadColor(0);
gStyle->SetPadBorderMode(0);
gStyle->SetCanvasBorderMode(0);
gStyle->SetFrameBorderMode(0);
gStyle->SetFillColor(0);
gStyle->SetLineColor(1);
gStyle->SetTitleTextColor(1);
gStyle->SetTitleColor(1);

TCanvas *cinv[2];
for(Int_t ip = 0; ip < 2; ip++)
{
cinv[ip] = new TCanvas(Form("cinv%d",ip),"cinv",10,10,600,600);
cinv[ip]->SetLeftMargin(0.2);
cinv[ip]->SetRightMargin(0.05);
cinv[ip]->SetBottomMargin(0.2);
}

TFile *f1 = new TFile("TMVApp_sig.root");
TH1F *hsig =(TH1F*)f1->Get("invsig");
TH1F *hsig1 =(TH1F*)f1->Get("invbkg");
hsig->Add(hsig1);
```

```
TH1F *htmva=(TH1F*)hsig->Clone("htmva");
htmva->Sumw2();

TFile *f2 = new TFile("TMVApp_bkg.root");
TH1F *hbkg =(TH1F*)f2->Get("invsig");
TH1F *hbkg1 =(TH1F*)f2->Get("invbkg");
hbkg->Add(hbkg1);
TH1F *htmva_bkg=(TH1F*)hbkg->Clone("htmva_bkg");
htmva_bkg->Sumw2();


htmva->GetXaxis()->SetTitle("M_{k#pi} (GeV/c^{2})");
htmva->GetXaxis()->SetTitleSize(0.05);
htmva->GetXaxis()->SetTitleOffset(1.03);
htmva->GetXaxis()->SetTitleFont(42);
htmva->GetXaxis()->CenterTitle(true);
htmva->GetXaxis()->SetLabelSize(0.05);
htmva->GetXaxis()->SetLabelFont(42);
htmva->GetXaxis()->SetNdivisions(511);
htmva->GetYaxis()->SetTitle("Counts(/10 MeV/c^{2})");
htmva->GetYaxis()->SetTitleFont(42);
htmva->GetYaxis()->CenterTitle(true);
htmva->GetYaxis()->SetTitleSize(0.05);
htmva->GetYaxis()->SetTitleOffset(1.7);
htmva->GetYaxis()->SetLabelSize(0.05);
htmva->GetYaxis()->SetLabelFont(42);
htmva->GetYaxis()->SetNdivisions(509);


htmva_bkg->GetXaxis()->SetTitle("M_{k#pi}");
htmva_bkg->GetXaxis()->SetTitleSize(0.05);
htmva_bkg->GetXaxis()->SetTitleOffset(1.03);
htmva_bkg->GetXaxis()->SetTitleFont(42);
htmva_bkg->GetXaxis()->CenterTitle(true);
htmva_bkg->GetXaxis()->SetLabelSize(0.05);
htmva_bkg->GetXaxis()->SetLabelFont(42);
htmva_bkg->GetXaxis()->SetNdivisions(511);
htmva_bkg->GetYaxis()->SetTitle("Counts(/10 MeV/c^{2})");
htmva_bkg->GetYaxis()->SetTitleFont(42);
htmva_bkg->GetYaxis()->CenterTitle(true);
htmva_bkg->GetYaxis()->SetTitleSize(0.05);
htmva_bkg->GetYaxis()->SetTitleOffset(1.7);
htmva_bkg->GetYaxis()->SetLabelSize(0.05);
htmva_bkg->GetYaxis()->SetLabelFont(42);
htmva_bkg->GetYaxis()->SetNdivisions(509);



htmva->SetMarkerStyle(20);
htmva->SetMarkerColor(1);
htmva->SetLineColor(1);
```

```
htmva_bkg->SetMarkerStyle(20);
htmva_bkg->SetMarkerColor(2);
htmva_bkg->SetLineColor(2);

TH1F *htmva_clone=(TH1F*)htmva->Clone("htmva_clone");
TH1F *htmva_bkg_clone=(TH1F*)htmva_bkg->Clone("htmva_bkg_clone");


float  fitlow =0.72;
float  fithi =1.2;
float  normlow=5;
float  normhigh=10;

htmva_bkg->Scale(htmva->
Integral(normlow,normhigh)/(htmva_bkg->Integral(normlow,normhigh)));
htmva_bkg_clone->Scale(htmva_clone->
Integral(normlow,normhigh)/(htmva_bkg_clone->Integral(normlow,normhigh)));


htmva->Add(htmva_bkg,-1);


TF1 *fitFcn1 = new TF1("fitFcn1",BWPoly2,fitlow,fithi,6);
fitFcn1->SetParameter(0,100);
fitFcn1->FixParameter(1,0.048);
fitFcn1->SetParLimits(2,0.85,0.95);
fitFcn1->SetParameter(3,10.0);
fitFcn1->SetParameter(4,100.0);
fitFcn1->SetParameter(5,10000.0);
fitFcn1->SetLineColor(2);
fitFcn1->SetParNames("Yield","Width","Mass","p0","p1","p2");

htmva->Fit(fitFcn1,"IER+");

float  a=fitFcn1->GetParameter(0);
float  b=fitFcn1->GetParameter(2);
float  c=fitFcn1->GetParameter(3);
float  d=fitFcn1->GetParameter(4);
float  e=fitFcn1->GetParameter(5);


TF1 *fitFcn2 = new TF1("fitFcn2",BW,fitlow,fithi,3);
fitFcn2->FixParameter(0,a);
fitFcn2->FixParameter(1,0.048);
fitFcn2->FixParameter(2,b);
fitFcn2->SetLineColor(93);
fitFcn2->SetLineStyle(2);


TF1 *fitFcn3 = new TF1("fitFcn3",Pol2,fitlow,fithi,3);
```

```
fitFcn3->FixParameter(0,c);
fitFcn3->FixParameter(1,d);
fitFcn3->FixParameter(2,e);
fitFcn3->SetLineColor(4);

// actual =34253

cout<<"signal by BW "<<fitFcn2->Integral(0.6,2.0)/0.01<<"\n";
cout<<"counts---"<<htmva_clone->Integral(10,50)<<"\n";
cout<<"Significance===="
<<(fitFcn2->Integral(0.6,2.0)/0.01)/sqrt(htmva_clone->Integral(10,50))<<"\n";


cinv[0]->cd();
htmva_clone->Draw();
htmva_bkg_clone->Draw("same");
TLegend *l1 = new TLegend(0.2365772,0.7801394,0.3708054,0.8797909);
l1->SetTextFont(42);
l1->SetBorderSize(0);
l1->SetFillStyle(0);
l1->SetFillColor(0);
l1->SetMargin(0.25);
l1->SetTextSize(0.03);
l1->SetEntrySeparation(0.5);
l1->AddEntry(htmva_clone,"Unlike pair","p");
l1->AddEntry(htmva_bkg_clone,"Normalized like pair","p");
l1->Draw();

cinv[1]->cd();
htmva->Draw();
fitFcn3->Draw("same");
fitFcn2->Draw("same");
TLegend *l2 = new TLegend(0.1565772,0.701394,0.508054,0.8797909);
l2->SetTextFont(42);
l2->SetBorderSize(0);
l2->SetFillStyle(0);
l2->SetFillColor(0);
l2->SetMargin(0.25);
l2->SetTextSize(0.03);
l2->SetEntrySeparation(0.5);
l2->AddEntry(fitFcn1,"BW + Pol2","l");
l2->AddEntry(fitFcn2,"BW","l");
l2->AddEntry(fitFcn3,"Pol2","l");
l2->Draw();


}
```

8.4 Scripts (SuperCDMS)

8.4.1 Extraction of Barium data.

```
/*
THIS CODE IS TO FILL RRQs USING Ba DATA.
data is in SLAC (NERO)
*/
#include "example_loadAnalysisFilesnewba.C" //Load Prodv5-6-3
#include "BasicCuts.h"                      //Load only Basic Cuts
#include <iostream>
#include <fstream>
#include <iomanip>
#include <TMath.h>
#include <iostream>
#include <fstream>

#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TFile.h>


void RRQ_fill_ba_ER(string weekno="1", Int_t zip=4)
{
TGaxis::SetMaxDigits(3);
gStyle->SetLabelSize(0.025, "Z");

//Define Filename variable, Not used anywhere, if unnecesary then discard this line later
string filename, fileaddress ;
filename="ba";
fileaddress ="ba";

//Convert weekno from string to int for wherever it might be convenient
Int_t wno= atoi(Form("%s", weekno.c_str()));

//Define source for proper naming of output file
string source, sourcefullname;


source="Ba";
sourcefullname="Barium";
// Initialize zlite
Int_t zlite ;
if (zip==4)
{ zlite =14;}
else
{
if (zip==14)
```

```
{ zlite =4;}
else
{
if (zip==5 || weekno==”Allz4” )
{
if (wno>=1 && wno<7)
zlite =14; cout<<”zlite: ”<< zlite<< endl;

}
else
{
if (zip==5 || weekno==”Allz14”)
{
if (wno>6 && wno<=12)
zlite =4; cout<<”zlite: ”<< zlite<< endl;
}
else
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}
}
}
}

// Initialize  weekno2
string weekno2=weekno;
if (weekno==”All” || weekno==”Allz4” || weekno==”Allz14”)
{
weekno=”∗”;
}

if (weekno2==”Allz4” && zip==4) //Added for convenience in running shell script
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}

if (weekno2==”Allz14” && zip==14) //Added for convenience in running shell script
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}

// Initialize  det
string det;
if (zip==4)
{det=”T2Z1”;}
```

```
else
{
if (zip==14)
{det="T5Z2";}
else
{
if (zip==5)
{det="T2Z2";}
else
{cout<<"Please enter either zip 4, 5 or 14";

}
}
}
// Initialize weektitle, Not used in this script but maybe useful in others
string weektitle;
if (weekno2=="All" && det=="T2Z1")
{
weektitle="Weeks1-6";
}
else
{

if (weekno2=="All" && det=="T5Z2")
{
weektitle="Weeks7-12";
}
else
{
if (det=="T2Z2" && weekno2=="Allz4")
{weektitle="Weeks5-6";}
else
{
if (det=="T2Z2" && weekno2=="Allz14")
{weektitle="Weeks7-12";}
else
{
if (source=="Sb" && weekno2=="All")
{//For Sb
weektitle="Weeks1-9";
}
else
{weektitle="Week";
weektitle=weektitle+weekno2;
}
}
}
}
}
```

```
//--------- Output File Address ---------
//string opfileaddress=Form("/home/viraj/Documents/DarkMatter/HT_Analysis/mf%s",
    det.c_str());
string opfileaddress="/home/u1/rik/Viraj/ba_bkg"; // Nero

//Setting up detector indices
Int_t DetIndex = zip-1;
Int_t tid = (DetIndex/3)+1;
Int_t zid = (DetIndex%3)+1;
cout<<"\nWe are looking at: T"<<tid<<"Z"<<zid<<endl;

//Confirming BiasFlashtime
//cout<<"BiasFlashTime= "<<flashtime()<<endl;

// ---------- Location of Data ----------
// --- Loading Prodv5-6-3 data ---
//string dataDir = "/home/viraj/Documents/DarkMatter/HT_Analysis/";
string dataDir = "/data/R134/dataReleases/Prodv5-3-5/merged/all/";

string fileaddress2 ;
fileaddress2 =fileaddress;
dataDir = dataDir+fileaddress2;
string seriesweek="week";
seriesweek=seriesweek+weekno;

//Call function that loads data
TChain *chain2 = chainDataAllSpecial(zip, dataDir, weekno, fileaddress, zlite );
//chain2->Scan();
cout<<"Number of events: "<<chain2->GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2->GetName()<<endl<<endl;

//Define Variables
double ptNFrange_1=-10; // Prev: -10, 0
double ptNFrange_2=170; // Prev: 170, 30
double ptNFbinsize=450; // Prev: 450, 300
double qbinsize=500; // Prev: 500, 160
double q_y_range_1=-10; //Prev: -10, -2
double q_y_range_2= 80; //Prev: 80, 14

const double binw= (ptNFrange_2-ptNFrange_1)/ptNFbinsize;


//Define some Quality cuts
TCut cLEChisq("PTNFchisq < (0.015*ptNF*ptNF+4500) && PTNFchisq >3600");
TCut cDeltaChisqGlitch("(PTOFchisq-PTglitch1OFchisq)<((-2.3*(ptNF*ptNF))+25)");
TCut cDeltaChiSqLFNLine("(PTOFchisq-PTlfnoise1OFchisq)< 14.2");
TCut cDeltaChiSqLFNParabola("(PTOFchisq-PTlfnoise1OFchisq)<
((-13.5152*(ptNF*ptNF))-(2.18824*ptNF)+27.692)");
TCut cDeltaChiSqLFN=cDeltaChiSqLFNLine+cDeltaChiSqLFNParabola;
TCut cChargeChiSqS1("QS1OFchisq <= ((( 0.00578106)*qsum1OF*qsum1OF*qsum1OF)
```

```
−( 0.427417∗qsum1OF∗qsum1OF)+( 9.88895∗qsum1OF)+ 5448.68)");
TCut cChargeChiSqS2("QS2OFchisq <= ((( 0.00327275)∗qsum2OF∗qsum2OF∗qsum2OF)
−( 0.25359∗qsum2OF∗qsum2OF)+( 7.05146∗qsum2OF)+ 6241.38)");
TCut cChargeChiSq=cChargeChiSqS1+cChargeChiSqS2;

TCut cRadialCutS1("qo1OF<3.0 || qi1OF>3.0");
TCut cRadialCutS2("qo2OF<1.8 || qi2OF>3.0");
TCut cRadialCut=cRadialCutS1+cRadialCutS2;

TCut cFVPolyUp("qi2OF<(−0.00233946∗qi1OF∗qi1OF)+1.11534∗qi1OF+2.06405");
TCut cFVPolyLow("qi2OF>(−0.0035754∗qi1OF∗qi1OF)+1.04275∗qi1OF−2.85197");
TCut cFVPoly=cFVPolyUp+cFVPolyLow;
TCut cHorizontalLine("qi2OF<3.5"); //mu+5.5∗sigma of qi2OF
TCut cVerticalLine("qi1OF<3.5"); // mu+13∗sigma of qi1OF
TCut cFlatLines=cHorizontalLine+cVerticalLine;
TCut cSymmetricFV=(cFVPoly)||(cFlatLines);

TCut cZeroCharge("((qi1OF+qi2OF)/2)<1 && ((qi1OF+qi2OF)/2)>−1");
TCut cPhononRadial("prpartOF<(TMath::Exp(−2.42429−0.0990673∗ptNF)+0.241877)"); // 2
    sigma
TCut cNotPhononRadial("prpartOF>(TMath::Exp(−2.42429−0.0990673∗ptNF)+0.241877)"); //
    2 sigma

//TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > (70/170)∗ptNF −10"); // simulation for 3
    sigma NR band cut
TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > 0.95∗precoiltNF −10 && ((qi1OF+qi2OF)/2.0)
    >3 "); // simulation for 3 sigma NR band cut with ptNF on x axis

TCut
    cQualityCuts=cLEChisq+cDeltaChisqGlitch+cDeltaChiSqLFN+cChargeChiSq+cSymmetricFV;



cout<<"\nNumber of events: "<<chain2−>GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2−>GetName()<<endl;

//chain2−>Scan("SeriesNumber","SeriesNumber==11509070319", "colsize=15 precision=13");



double livetime;
double Empty, EventCategory, volt;
double livetimesec;
double netlivetime = 0.0;
//chain2−>SetBranchStatus("∗",0);

//reactivate only the branches we need before setting branch adress
//chain2−>SetBranchStatus("LiveTime",1);
chain2−>SetBranchAddress("LiveTime", &livetime);
```

```
double precoiltNF_local,ytNF_local,
    qzpartOF_local,qrpart1OF_local,pzpartOF_local,prpartOF_local,qsummaxOF_local;

double PTNFchisq_local,
ptNF_local,
PTOFchisq_local,
PTglitch1OFchisq_local,
PTlfnoise1OFchisq_local,
QS1OFchisq_local,
qsum1OF_local,
QS2OFchisq_local,
qsum2OF_local,
qo1OF_local,
qo2OF_local,
qi1OF_local,
qi2OF_local;


chain2->SetBranchAddress("PTNFchisq",&PTNFchisq_local);
chain2->SetBranchAddress("ptNF",&ptNF_local);
chain2->SetBranchAddress("PTOFchisq",&PTOFchisq_local);
chain2->SetBranchAddress("PTglitch1OFchisq",&PTglitch1OFchisq_local);
chain2->SetBranchAddress("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_local);
chain2->SetBranchAddress("QS1OFchisq",&QS1OFchisq_local);
chain2->SetBranchAddress("qsum1OF",&qsum1OF_local);
chain2->SetBranchAddress("QS2OFchisq",&QS2OFchisq_local);
chain2->SetBranchAddress("qsum2OF",&qsum2OF_local);
chain2->SetBranchAddress("qo1OF",&qo1OF_local);
chain2->SetBranchAddress("qo2OF",&qo2OF_local);
chain2->SetBranchAddress("qi1OF",&qi1OF_local);
chain2->SetBranchAddress("qi2OF",&qi2OF_local);



chain2->SetBranchAddress("precoiltNF",&precoiltNF_local);
chain2->SetBranchAddress("ytNF", &ytNF_local);
chain2->SetBranchAddress("qzpartOF", &qzpartOF_local);
chain2->SetBranchAddress("qrpart1OF", &qrpart1OF_local);
chain2->SetBranchAddress("pzpartOF", &pzpartOF_local);
chain2->SetBranchAddress("prpartOF", &prpartOF_local);
chain2->SetBranchAddress("qsummaxOF", &qsummaxOF_local);
chain2->SetBranchAddress("EventCategory", &EventCategory);


int  ctr  = 0;
double flashtime;
//chain2->SetBranchStatus("BiasFlashTime",1);
chain2->SetBranchAddress("BiasFlashTime", &flashtime);

float  precoiltNF_b,ytNF_b, qzpartOF_b,qrpart1OF_b,pzpartOF_b,prpartOF_b,qsummaxOF_b;
float  PTNFchisq_b,ptNF_b,PTOFchisq_b,PTglitch1OFchisq_b,PTlfnoise1OFchisq_b,
```

```
QS1OFchisq_b,qsum1OF_b,QS2OFchisq_b,qsum2OF_b,
qo1OF_b,qo2OF_b,qi1OF_b,qi2OF_b,qimean_b;


TFile *fout = new TFile("RRQ_ER.root","RECREATE");
TTree *tree_bkg = new TTree("tree_bkg","Bkg tree");
tree_bkg->Branch("precoiltNF",&precoiltNF_b,"precoiltNF_b/F");
tree_bkg->Branch("ytNF",&ytNF_b,"ytNF_b/F");
tree_bkg->Branch("qzpartOF",&qzpartOF_b,"qzpartOF_b/F");
tree_bkg->Branch("qrpart1OF",&qrpart1OF_b,"qrpart1OF_b/F");
tree_bkg->Branch("pzpartOF",&pzpartOF_b,"pzpartOF_b/F");
tree_bkg->Branch("prpartOF",&prpartOF_b,"prpartOF_b/F");
tree_bkg->Branch("qsummaxOF",&qsummaxOF_b,"qsummaxOF_b/F");

tree_bkg->Branch("PTNFchisq",&PTNFchisq_b,"PTNFchisq_b/F");
tree_bkg->Branch("ptNF",&ptNF_b,"ptNF_b/F");
tree_bkg->Branch("PTOFchisq",&PTOFchisq_b,"PTOFchisq_b/F");
tree_bkg->Branch("PTglitch1OFchisq",&PTglitch1OFchisq_b,"PTglitch1OFchisq_b/F");
tree_bkg->Branch("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_b,"PTlfnoise1OFchisq_b/F");
tree_bkg->Branch("QS1OFchisq",&QS1OFchisq_b,"QS1OFchisq_b/F");
tree_bkg->Branch("qsum1OF",&qsum1OF_b,"qsum1OF_b/F");
tree_bkg->Branch("QS2OFchisq",&QS2OFchisq_b,"QS2OFchisq_b/F");
tree_bkg->Branch("qsum2OF",&qsum2OF_b,"qsum2OF_b/F");
tree_bkg->Branch("qo1OF",&qo1OF_b,"qo1OF_b/F");
tree_bkg->Branch("qo2OF",&qo2OF_b,"qo2OF_b/F");
tree_bkg->Branch("qi1OF",&qi1OF_b,"qi1OF_b/F");
tree_bkg->Branch("qi2OF",&qi2OF_b,"qi2OF_b/F");
tree_bkg->Branch("qimean",&qimean_b,"qimean_b/F");


int   filledcount ;
 filledcount =0;


double precoiltNFrange_1=10;
double precoiltNFrange_2=170;
double precoiltNFbinsize=8;


double qzpartOFrange_1=-0.3;
double qzpartOFrange_2=0.3;


double qrpart1OFrange_1=-0.2;
double qrpart1OFrange_2=0.6;


double ytNFrange_1=0.6;
double ytNFrange_2=1.4;
```

```
double prpartOFrange_1=0.1;
double prpartOFrange_2=0.3;


double pzpartOFrange_1=−0.1;
double pzpartOFrange_2=0.2;

double qsummaxOFrange_1=0;
double qsummaxOFrange_2=150;

/*TH1D *h_precoiltNF = new TH1D("h_precoiltNF", Form("T%dZ%d: %s−%s, %s,
    precoiltNF", tid, zid, sourcefullname.c_str(), filename.c_str(), weektitle.c_str()),
    precoiltNFbinsize, precoiltNFrange_1, precoiltNFrange_2);
h_precoiltNF−>SetXTitle("precoiltNF [keV]");
h_precoiltNF−>SetYTitle("Counts");
h_precoiltNF−>GetYaxis()−>SetTitleOffset(1.5);
h_precoiltNF−>GetYaxis()−>SetLabelSize(0.03);
h_precoiltNF−>GetYaxis()−>SetTitleSize(0.03);
*/

/*double ptNFrange_1=−10; // Prev: −10, 0
double ptNFrange_2=170; // Prev: 170, 30
double ptNFbinsize=450; // Prev: 450, 300
double qbinsize=500; // Prev: 500, 160
double q_y_range_1=−10; //Prev: −10, −2
double q_y_range_2= 80; //Prev: 80, 14

const double binw= (ptNFrange_2−ptNFrange_1)/ptNFbinsize;
double qimean=0;

TCanvas* qptNF= new TCanvas("qptNF", "qimean vs precoiltNF (RRQ_fill script)", 600, 600);
TH2D *h_ERNRallcuts = new TH2D("h_ERNRallcuts", Form("T%dZ%d: %s−%s, %s,
    Ionization vs precoiltNF", tid, zid, sourcefullname.c_str(), filename.c_str(), weektitle.c_str()),
    ptNFbinsize, ptNFrange_1, ptNFrange_2, qbinsize, q_y_range_1, q_y_range_2);
h_ERNRallcuts−>SetXTitle("precoiltNF [keV]");
h_ERNRallcuts−>SetYTitle("qimean");
h_ERNRallcuts−>GetYaxis()−>SetTitleOffset(1.5);
h_ERNRallcuts−>GetYaxis()−>SetLabelSize(0.03);
h_ERNRallcuts−>GetYaxis()−>SetTitleSize(0.03);
*/
//TCanvas* c_precoiltNF= new TCanvas("c_precoiltNF", "c_precoiltNF (All Cuts) 1", 600, 600);
cout<<"start while loop"<<endl;
//while(chain2−>GetEntry(ctr))
while(chain2−>GetEntry(ctr))
{
// cout<<"outside if condition "<<"precoiltNF: "<<precoiltNF_local<<endl;
 if(ctr%50000==0)cout<<"Number of events processed ========== "<<ctr<<endl;
//  if(/*flashtime  < 3300.00 && EventCategory!=1 && */ precoiltNF_local>10)//T2Z1: 3300 ,
    T5Z2: 10800

//   if(PTNFchisq_local < (0.015*ptNF_local*ptNF_local+4500) && PTNFchisq_local >3600
```

```
//    && (PTOFchisq_local−PTglitch1OFchisq_local)<((−2.3∗(ptNF_local∗ptNF_local))+25)
//   && (PTOFchisq_local−PTlfnoise1OFchisq_local)< 14.2
//   && (PTOFchisq_local−PTlfnoise1OFchisq_local)<((−13.5152∗(ptNF_local∗ptNF_local))
//−(2.18824∗ptNF_local)+27.692)
//   && QS1OFchisq_local <= ((( 0.00578106)∗qsum1OF_local∗qsum1OF_local∗qsum1OF_local)−
//(0.427417∗qsum1OF_local∗qsum1OF_local)+( 9.88895∗qsum1OF_local)+ 5448.68)
//   && QS2OFchisq_local <= ((( //0.00327275)∗qsum2OF_local∗qsum2OF_local∗qsum2OF_local)
//−( 0.25359∗qsum2OF_local∗qsum2OF_local)+( 7.05146∗qsum2OF_local)+ 6241.38)
//   && (qo1OF_local<3.0 || qi1OF_local>3.0) && (qo2OF_local<1.8 || qi2OF_local>3.0)
//&& ( (/∗parallel∗/
      qi2OF_local<(−0.00233946∗qi1OF_local∗qi1OF_local)+1.11534∗qi1OF_local+2.06405 &&
      qi2OF_local>(−0.0035754∗qi1OF_local∗qi1OF_local)+1.04275∗qi1OF_local−2.85197 )

//||  (qi2OF_local<3.5 && qi1OF_local<3.5) /∗cSymmetricFV∗/)

// && ( (    ((qi1OF_local+qi2OF_local)/2.0 ) > 0.95∗precoiltNF_b −10 ) &&
      ((qi1OF_local+qi2OF_local)/2.0) >3 )
// ) //if  cuts  ends
// {
if (precoiltNF_local>precoiltNFrange_1 && precoiltNF_local<precoiltNFrange_2 &&
      ((qi1OF_local+qi2OF_local)/2.0 ) < 160
&& ytNF_local>0 && ytNF_local<2
&& qzpartOF_local>qzpartOFrange_1 && qzpartOF_local< qzpartOFrange_2 &&
      qrpart1OF_local> qrpart1OFrange_1 && qrpart1OF_local < qrpart1OFrange_2
&& prpartOF_local> prpartOFrange_1 && prpartOF_local < prpartOFrange_2 &&
      pzpartOF_local>pzpartOFrange_1 && pzpartOF_local<pzpartOFrange_2
//&& qsummaxOF_local > qsummaxOFrange_1 && qsummaxOF_local < qsummaxOFrange_2
)// for  range of  features  T2Z1: 3300 , T5Z2: 10800
{
 filledcount ++;
//   cout<<"livetime : "<<livetime<<endl;
//    cout<<" precoiltNF : "<<precoiltNF_local<<endl;
//   h_precoiltNF−>Fill(precoiltNF_local);

precoiltNF_b=precoiltNF_local;
ytNF_b=ytNF_local;
qzpartOF_b=qzpartOF_local;
qrpart1OF_b=qrpart1OF_local;
pzpartOF_b=pzpartOF_local;
prpartOF_b=prpartOF_local;
qsummaxOF_b=qsummaxOF_local;

PTNFchisq_b=PTNFchisq_local;
ptNF_b=ptNF_local;
PTOFchisq_b=PTOFchisq_local;
PTglitch1OFchisq_b=PTglitch1OFchisq_local;
PTlfnoise1OFchisq_b=PTlfnoise1OFchisq_local;
QS1OFchisq_b=QS1OFchisq_local;
qsum1OF_b=qsum1OF_local;
QS2OFchisq_b=QS2OFchisq_local;
qsum2OF_b=qsum2OF_local;
```

```
qo1OF_b=qo1OF_local;
qo2OF_b=qo2OF_local;
qi1OF_b=qi1OF_local;
qi2OF_b=qi2OF_local;
qimean_b=0.5*(qi1OF_local+qi2OF_local);


//qimean = 0.5*( qi1OF_local+ qi2OF_local );
//h_ERNRallcuts->Fill( precoiltNF_local , qimean );
tree_bkg->Fill();
}  // range of  variables  IF

//  } // TCut IF

ctr++;
//  if(ctr>1000000) break;

} // while loop ends
//break;
//h_precoiltNF->Draw();
//cout<<"h integral :"<<h_precoiltNF->Integral()<<endl;
/*TLegend *leg=new TLegend(0.55,0.57,0.90,0.77);
leg->SetBorderSize(0);
leg->AddEntry(h_precoiltNF,"after Quality+PRC cuts+ 3sigma NR cut","lpf");
leg->Draw("same");

c_precoiltNF->Update();
*/

cout<<"ctr (counts) "<<ctr<<endl;
cout<<"filled (counts) "<<filledcount<<endl;

fout->cd();
tree_bkg->Scan();
tree_bkg->Write();
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("precoiltNF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("ytNF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qzpartOF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qrpart1OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("pzpartOF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("prpartOF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qsummaxOF"));


tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTNFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("ptNF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTOFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTglitch1OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTlfnoise1OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("QS1OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qsum1OF"));
```

```
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("QS2OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qsum2OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qo1OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qo2OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qi1OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qi2OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qimean"));


tree_bkg->Scan();
fout->Close();

/*
Double_t ylinefun(double x)
{
return 0.95*x + 10 ; }
const double m= 0.95; //slope
const double c=-10; // y intecept

//TLine *line1 = new TLine(0,-10,170,60);
//TLine *line1 = new TLine((3-c)/m,3,80,ylinefun(80));
// line1->SetLineWidth(4);
// line1->SetLineColor(2);
//line1->Draw("same");
//TLine *line2 = new TLine(0,3,(3-c)/m,3); // y=3 line
//   line2->SetLineWidth(4);
//line2->SetLineColor(2);
//line2->Draw("same");
h_ERNRallcuts->Print();


qptNF->Update();

// Save the output plot in .gif form
qptNF->SaveAs(Form("qiprecoiltNF_RRQfill_%s_%s_%s.gif",
     fileaddress.c_str(),det.c_str(),weektitle.c_str()));
*/



}
```

8.4.2 Extraction of Californium data.

```
/*
THIS CODE IS TO Get RRQs USING Cf DATA.
data is in SLAC (nero)
*/
#include "example_loadAnalysisFilesnewcf.C" //Load Prodv5-6-3
#include "BasicCuts.h"                      //Load only Basic Cuts
#include <iostream>
#include <fstream>
#include <iomanip>
#include <TMath.h>
#include <iostream>
#include <fstream>

#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TFile.h>


void RRQ_fill_cf(string weekno="1", Int_t zip=4)
{
TGaxis::SetMaxDigits(3);
gStyle->SetLabelSize(0.025, "Z");

//Define Filename variable, Not used anywhere, if unnecesary then discard this line later
string filename, fileaddress ;
filename="cf";
fileaddress ="cf";

//Convert weekno from string to int for wherever it might be convenient
Int_t wno= atoi(Form("%s", weekno.c_str()));

//Define source for proper naming of output file
string source, sourcefullname;


source="cf";
sourcefullname="Californium";
// Initialize zlite
Int_t zlite ;
if(zip==4)
{zlite=14;}
else
{
if(zip==14)
{zlite=4;}
else
{
```

```
if (zip==5 || weekno==”Allz4” )
{
if (wno>=1 && wno<7)
zlite =14; cout<<”zlite: ”<< zlite<< endl;

}
else
{
if (zip==5 || weekno==”Allz14”)
{
if (wno>6 && wno<=12)
zlite =4; cout<<”zlite: ”<< zlite<< endl;
}
else
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}
}
}
}

// Initialize  weekno2
string weekno2=weekno;
if (weekno==”All” || weekno==”Allz4” || weekno==”Allz14”)
{
weekno=”∗”;
}

if (weekno2==”Allz4” && zip==4) //Added for convenience in running shell script
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}

if (weekno2==”Allz14” && zip==14) //Added for convenience in running shell script
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}

// Initialize  det
string det;
if (zip==4)
{det=”T2Z1”;}
else
{
if (zip==14)
```

```
{det="T5Z2";}
else
{
if(zip==5)
{det="T2Z2";}
else
{cout<<"Please enter either zip 4, 5 or 14";

}
}
}
// Initialize weektitle, Not used in this script but maybe useful in others
string weektitle;
if(weekno2=="All" && det=="T2Z1")
{
weektitle="Weeks1-6";
}
else
{

if(weekno2=="All" && det=="T5Z2")
{
weektitle="Weeks7-12";
}
else
{
if(det=="T2Z2" && weekno2=="Allz4")
{weektitle="Weeks5-6";}
else
{
if(det=="T2Z2" && weekno2=="Allz14")
{weektitle="Weeks7-12";}
else
{
if(source=="Sb" && weekno2=="All")
{//For Sb
weektitle="Weeks1-9";
}
else
{weektitle="Week";
weektitle=weektitle+weekno2;
}
}
}
}
}



//------- Output File Address --------
```

```
//string opfileaddress=Form("/home/viraj/Documents/DarkMatter/HT_Analysis/mf%s",
    det.c_str());
string opfileaddress="/home/u1/rik/Viraj/cf";

//Setting up detector indices
Int_t DetIndex = zip-1;
Int_t tid = (DetIndex/3)+1;
Int_t zid = (DetIndex%3)+1;
cout<<"\nWe are looking at: T"<<tid<<"Z"<<zid<<endl;

//Confirming BiasFlashtime
//cout<<"BiasFlashTime= "<<flashtime()<<endl;

// --------- Location of Data --------
// --- Loading Prodv5-6-3 data ---
//string dataDir = "/home/viraj/Documents/DarkMatter/HT_Analysis/";
string dataDir = "/data/R134/dataReleases/Prodv5-3-5/merged/all/";

string fileaddress2;
fileaddress2=fileaddress;
dataDir = dataDir+fileaddress2;
string seriesweek="week";
seriesweek=seriesweek+weekno;

//Call function that loads data
TChain *chain2 = chainDataAllSpecial(zip, dataDir, weekno, fileaddress, zlite);
//chain2->Scan();
cout<<"Number of events: "<<chain2->GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2->GetName()<<endl<<endl;

//Define Variables
double ptNFrange_1=-10; // Prev: -10, 0
double ptNFrange_2=170; // Prev: 170, 30
double ptNFbinsize=450; // Prev: 450, 300
double qbinsize=500; // Prev: 500, 160
double q_y_range_1=-10; //Prev: -10, -2
double q_y_range_2= 80; //Prev: 80, 14

const double binw= (ptNFrange_2-ptNFrange_1)/ptNFbinsize;


//Define some Quality cuts
TCut cLEChisq("PTNFchisq < (0.015*ptNF*ptNF+4500) && PTNFchisq >3600");
TCut cDeltaChisqGlitch("(PTOFchisq-PTglitch1OFchisq)<((-2.3*(ptNF*ptNF))+25)");
TCut cDeltaChiSqLFNLine("(PTOFchisq-PTlfnoise1OFchisq)< 14.2");
TCut cDeltaChiSqLFNParabola("(PTOFchisq-PTlfnoise1OFchisq)<
((-13.5152*(ptNF*ptNF))-(2.18824*ptNF)+27.692)");
TCut cDeltaChiSqLFN=cDeltaChiSqLFNLine+cDeltaChiSqLFNParabola;
TCut cChargeChiSqS1("QS1OFchisq <= ((( 0.00578106)*qsum1OF*qsum1OF*qsum1OF)
-( 0.427417*qsum1OF*qsum1OF)+( 9.88895*qsum1OF)+ 5448.68)");
TCut cChargeChiSqS2("QS2OFchisq <= ((( 0.00327275)*qsum2OF*qsum2OF*qsum2OF)
```

```
−( 0.25359∗qsum2OF∗qsum2OF)+( 7.05146∗qsum2OF)+ 6241.38)");
TCut cChargeChiSq=cChargeChiSqS1+cChargeChiSqS2;

TCut cRadialCutS1("qo1OF<3.0 || qi1OF>3.0");
TCut cRadialCutS2("qo2OF<1.8 || qi2OF>3.0");
TCut cRadialCut=cRadialCutS1+cRadialCutS2;

TCut cFVPolyUp("qi2OF<(−0.00233946∗qi1OF∗qi1OF)+1.11534∗qi1OF+2.06405");
TCut cFVPolyLow("qi2OF>(−0.0035754∗qi1OF∗qi1OF)+1.04275∗qi1OF−2.85197");
TCut cFVPoly=cFVPolyUp+cFVPolyLow;
TCut cHorizontalLine("qi2OF<3.5"); //mu+5.5∗sigma of qi2OF
TCut cVerticalLine("qi1OF<3.5"); // mu+13∗sigma of qi1OF
TCut cFlatLines=cHorizontalLine+cVerticalLine;
TCut cSymmetricFV=(cFVPoly)||(cFlatLines);

TCut cZeroCharge("((qi1OF+qi2OF)/2)<1 && ((qi1OF+qi2OF)/2)>−1");
TCut cPhononRadial("prpartOF<(TMath::Exp(−2.42429−0.0990673∗ptNF)+0.241877)");
// 2 sigma
TCut cNotPhononRadial("prpartOF>(TMath::Exp(−2.42429−0.0990673∗ptNF)+0.241877)");
// 2 sigma

//TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > (70/170)∗ptNF −10"); // simulation for 3
    sigma NR band cut
TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > 0.95∗precoiltNF −10 && ((qi1OF+qi2OF)/2.0)
    >3 "); // simulation for 3 sigma NR band cut with ptNF on x axis

TCut cQualityCuts=
cLEChisq+cDeltaChisqGlitch+cDeltaChiSqLFN+cChargeChiSq+cSymmetricFV;



cout<<"\nNumber of events: "<<chain2−>GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2−>GetName()<<endl;

//chain2−>Scan("SeriesNumber","SeriesNumber==11509070319", "colsize=15 precision=13");



double livetime;
double Empty, EventCategory, volt;
double livetimesec;
double netlivetime = 0.0;
//chain2−>SetBranchStatus("∗",0);

//reactivate only the branches we need before setting branch adress
//chain2−>SetBranchStatus("LiveTime",1);
chain2−>SetBranchAddress("LiveTime", &livetime);


double precoiltNF_local,ytNF_local,
    qzpartOF_local,qrpart1OF_local,pzpartOF_local,prpartOF_local,qsummaxOF_local;
```

```
double PTNFchisq_local,
ptNF_local,
PTOFchisq_local,
PTglitch1OFchisq_local,
PTlfnoise1OFchisq_local,
QS1OFchisq_local,
qsum1OF_local,
QS2OFchisq_local,
qsum2OF_local,
qo1OF_local,
qo2OF_local,
qi1OF_local,
qi2OF_local;


chain2->SetBranchAddress("PTNFchisq",&PTNFchisq_local);
chain2->SetBranchAddress("ptNF",&ptNF_local);
chain2->SetBranchAddress("PTOFchisq",&PTOFchisq_local);
chain2->SetBranchAddress("PTglitch1OFchisq",&PTglitch1OFchisq_local);
chain2->SetBranchAddress("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_local);
chain2->SetBranchAddress("QS1OFchisq",&QS1OFchisq_local);
chain2->SetBranchAddress("qsum1OF",&qsum1OF_local);
chain2->SetBranchAddress("QS2OFchisq",&QS2OFchisq_local);
chain2->SetBranchAddress("qsum2OF",&qsum2OF_local);
chain2->SetBranchAddress("qo1OF",&qo1OF_local);
chain2->SetBranchAddress("qo2OF",&qo2OF_local);
chain2->SetBranchAddress("qi1OF",&qi1OF_local);
chain2->SetBranchAddress("qi2OF",&qi2OF_local);



chain2->SetBranchAddress("precoiltNF",&precoiltNF_local);
chain2->SetBranchAddress("ytNF", &ytNF_local);
chain2->SetBranchAddress("qzpartOF", &qzpartOF_local);
chain2->SetBranchAddress("qrpart1OF", &qrpart1OF_local);
chain2->SetBranchAddress("pzpartOF", &pzpartOF_local);
chain2->SetBranchAddress("prpartOF", &prpartOF_local);
chain2->SetBranchAddress("qsummaxOF", &qsummaxOF_local);
chain2->SetBranchAddress("EventCategory", &EventCategory);


int  ctr  = 0;
double flashtime;
//chain2->SetBranchStatus("BiasFlashTime",1);
chain2->SetBranchAddress("BiasFlashTime", &flashtime);

float  precoiltNF_b,ytNF_b, qzpartOF_b,qrpart1OF_b,pzpartOF_b,prpartOF_b,qsummaxOF_b;
float  PTNFchisq_b,ptNF_b,PTOFchisq_b,PTglitch1OFchisq_b,PTlfnoise1OFchisq_b,
QS1OFchisq_b,qsum1OF_b,QS2OFchisq_b,qsum2OF_b,
qo1OF_b,qo2OF_b,qi1OF_b,qi2OF_b,qimean_b;
```

```
TFile *fout = new TFile("RRQ_NR_cf.root","RECREATE");
TTree *tree_bkg = new TTree("tree_bkg","Bkg tree");
tree_bkg->Branch("precoiltNF",&precoiltNF_b,"precoiltNF_b/F");
tree_bkg->Branch("ytNF",&ytNF_b,"ytNF_b/F");
tree_bkg->Branch("qzpartOF",&qzpartOF_b,"qzpartOF_b/F");
tree_bkg->Branch("qrpart1OF",&qrpart1OF_b,"qrpart1OF_b/F");
tree_bkg->Branch("pzpartOF",&pzpartOF_b,"pzpartOF_b/F");
tree_bkg->Branch("prpartOF",&prpartOF_b,"prpartOF_b/F");
tree_bkg->Branch("qsummaxOF",&qsummaxOF_b,"qsummaxOF_b/F");

tree_bkg->Branch("PTNFchisq",&PTNFchisq_b,"PTNFchisq_b/F");
tree_bkg->Branch("ptNF",&ptNF_b,"ptNF_b/F");
tree_bkg->Branch("PTOFchisq",&PTOFchisq_b,"PTOFchisq_b/F");
tree_bkg->Branch("PTglitch1OFchisq",&PTglitch1OFchisq_b,"PTglitch1OFchisq_b/F");
tree_bkg->Branch("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_b,"PTlfnoise1OFchisq_b/F");
tree_bkg->Branch("QS1OFchisq",&QS1OFchisq_b,"QS1OFchisq_b/F");
tree_bkg->Branch("qsum1OF",&qsum1OF_b,"qsum1OF_b/F");
tree_bkg->Branch("QS2OFchisq",&QS2OFchisq_b,"QS2OFchisq_b/F");
tree_bkg->Branch("qsum2OF",&qsum2OF_b,"qsum2OF_b/F");
tree_bkg->Branch("qo1OF",&qo1OF_b,"qo1OF_b/F");
tree_bkg->Branch("qo2OF",&qo2OF_b,"qo2OF_b/F");
tree_bkg->Branch("qi1OF",&qi1OF_b,"qi1OF_b/F");
tree_bkg->Branch("qi2OF",&qi2OF_b,"qi2OF_b/F");
tree_bkg->Branch("qimean",&qimean_b,"qimean_b/F");




int   filledcount ;
 filledcount =0;


double precoiltNFrange_1=10;
double precoiltNFrange_2=170;
double precoiltNFbinsize=8;


double qzpartOFrange_1=-0.3;
double qzpartOFrange_2=0.3;


double qrpart1OFrange_1=-0.2;
double qrpart1OFrange_2=0.6;


double ytNFrange_1=0.6;
double ytNFrange_2=1.4;
```

```
double prpartOFrange_1=0.1;
double prpartOFrange_2=0.3;


double pzpartOFrange_1=−0.1;
double pzpartOFrange_2=0.2;

double qsummaxOFrange_1=0;
double qsummaxOFrange_2=150;

/*TH1D *h_precoiltNF = new TH1D("h_precoiltNF", Form("T%dZ%d: %s−%s, %s,
    precoiltNF", tid, zid, sourcefullname.c_str(), filename.c_str(), weektitle.c_str()),
    precoiltNFbinsize, precoiltNFrange_1, precoiltNFrange_2);
h_precoiltNF−>SetXTitle("precoiltNF [keV]");
h_precoiltNF−>SetYTitle("Counts");
h_precoiltNF−>GetYaxis()−>SetTitleOffset(1.5);
h_precoiltNF−>GetYaxis()−>SetLabelSize(0.03);
h_precoiltNF−>GetYaxis()−>SetTitleSize(0.03);
*/

/*double ptNFrange_1=−10; // Prev: −10, 0
double ptNFrange_2=170; // Prev: 170, 30
double ptNFbinsize=450; // Prev: 450, 300
double qbinsize=500; // Prev: 500, 160
double q_y_range_1=−10; //Prev: −10, −2
double q_y_range_2= 80; //Prev: 80, 14

const double binw= (ptNFrange_2−ptNFrange_1)/ptNFbinsize;
double qimean=0;

TCanvas* qptNF= new TCanvas("qptNF", "qimean vs precoiltNF (RRQ_fill script)", 600, 600);
TH2D *h_ERNRallcuts = new TH2D("h_ERNRallcuts", Form("T%dZ%d: %s−%s, %s,
    Ionization vs precoiltNF", tid, zid, sourcefullname.c_str(), filename.c_str(), weektitle.c_str()),
    ptNFbinsize, ptNFrange_1, ptNFrange_2, qbinsize, q_y_range_1, q_y_range_2);
h_ERNRallcuts−>SetXTitle("precoiltNF [keV]");
h_ERNRallcuts−>SetYTitle("qimean");
h_ERNRallcuts−>GetYaxis()−>SetTitleOffset(1.5);
h_ERNRallcuts−>GetYaxis()−>SetLabelSize(0.03);
h_ERNRallcuts−>GetYaxis()−>SetTitleSize(0.03);
*/
//TCanvas* c_precoiltNF= new TCanvas("c_precoiltNF", "c_precoiltNF (All Cuts) 1", 600, 600);
cout<<"start while loop"<<endl;
//while(chain2−>GetEntry(ctr))
while(chain2−>GetEntry(ctr))
{
// cout<<"outside if condition "<<"precoiltNF: "<<precoiltNF_local<<endl;
 if(ctr%50000==0)cout<<"Number of events processed ========== "<<ctr<<endl;
//  if(/*flashtime  < 3300.00 && EventCategory!=1 && */ precoiltNF_local>10)//T2Z1: 3300 ,
    T5Z2: 10800
```

```
//    if(PTNFchisq_local < (0.015*ptNF_local*ptNF_local+4500) && PTNFchisq_local >3600
//    && (PTOFchisq_local-PTglitch1OFchisq_local)<((-2.3*(ptNF_local*ptNF_local))+25)
//    && (PTOFchisq_local-PTlfnoise1OFchisq_local)< 14.2
//    && (PTOFchisq_local-PTlfnoise1OFchisq_local)<
//((-13.5152*(ptNF_local*ptNF_local))-(2.18824*ptNF_local)+27.692)
//     && QS1OFchisq_local <= (((
      0.00578106)*qsum1OF_local*qsum1OF_local*qsum1OF_local)-(
      0.427417*qsum1OF_local*qsum1OF_local)+( 9.88895*qsum1OF_local)+ 5448.68)
//     && QS2OFchisq_local <= (((
      0.00327275)*qsum2OF_local*qsum2OF_local*qsum2OF_local)-(
      0.25359*qsum2OF_local*qsum2OF_local)+( 7.05146*qsum2OF_local)+ 6241.38)
//     && (qo1OF_local<3.0 || qi1OF_local>3.0) && (qo2OF_local<1.8 || qi2OF_local>3.0)
//&& ( (*parallel*/
      qi2OF_local<(-0.00233946*qi1OF_local*qi1OF_local)+1.11534*qi1OF_local+2.06405 &&
      qi2OF_local>(-0.0035754*qi1OF_local*qi1OF_local)+1.04275*qi1OF_local-2.85197 )

//||  (qi2OF_local<3.5 && qi1OF_local<3.5) /*cSymmetricFV*/)

// && ( (   ((qi1OF_local+qi2OF_local)/2.0 ) > 0.95*precoiltNF_b -10 ) &&
      ((qi1OF_local+qi2OF_local)/2.0) >3 )
//   ) //if cuts ends
// {
if(precoiltNF_local>precoiltNFrange_1 && precoiltNF_local<precoiltNFrange_2 &&
      ((qi1OF_local+qi2OF_local)/2.0 ) < 160
&& ytNF_local>0 && ytNF_local<2
&& qzpartOF_local>qzpartOFrange_1 && qzpartOF_local< qzpartOFrange_2 &&
      qrpart1OF_local> qrpart1OFrange_1 && qrpart1OF_local < qrpart1OFrange_2
&& prpartOF_local> prpartOFrange_1 && prpartOF_local < prpartOFrange_2 &&
      pzpartOF_local>pzpartOFrange_1 && pzpartOF_local<pzpartOFrange_2
// && qsummaxOF_local > qsummaxOFrange_1 && qsummaxOF_local < qsummaxOFrange_2
)// for range of features T2Z1: 3300 , T5Z2: 10800
{
 filledcount ++;
//   cout<<"livetime : "<<livetime<<endl;
//    cout<<" precoiltNF : "<<precoiltNF_local<<endl;
//   h_precoiltNF->Fill(precoiltNF_local);

precoiltNF_b=precoiltNF_local;
ytNF_b=ytNF_local;
qzpartOF_b=qzpartOF_local;
qrpart1OF_b=qrpart1OF_local;
pzpartOF_b=pzpartOF_local;
prpartOF_b=prpartOF_local;
qsummaxOF_b=qsummaxOF_local;


PTNFchisq_b=PTNFchisq_local;
ptNF_b=ptNF_local;
PTOFchisq_b=PTOFchisq_local;
PTglitch1OFchisq_b=PTglitch1OFchisq_local;
PTlfnoise1OFchisq_b=PTlfnoise1OFchisq_local;
```

```
QS1OFchisq_b=QS1OFchisq_local;
qsum1OF_b=qsum1OF_local;
QS2OFchisq_b=QS2OFchisq_local;
qsum2OF_b=qsum2OF_local;
qo1OF_b=qo1OF_local;
qo2OF_b=qo2OF_local;
qi1OF_b=qi1OF_local;
qi2OF_b=qi2OF_local;
qimean_b=0.5*(qi1OF_local+qi2OF_local);


//qimean = 0.5*( qi1OF_local+ qi2OF_local );
//h_ERNRallcuts->Fill( precoiltNF_local , qimean );
tree_bkg->Fill();
}  // range of  variables  IF

// } // TCut IF

ctr++;
//   if(ctr>10000) break;

} // while loop ends
//break;
//h_precoiltNF->Draw();
//cout<<"h integral :"<<h_precoiltNF->Integral()<<endl;
/*TLegend *leg=new TLegend(0.55,0.57,0.90,0.77);
leg->SetBorderSize(0);
leg->AddEntry(h_precoiltNF,"after Quality+PRC cuts+ 3sigma NR cut","lpf");
leg->Draw("same");

c_precoiltNF->Update();
*/

cout<<"ctr (counts) "<<ctr<<endl;
cout<<"filled (counts) "<<filledcount<<endl;

fout->cd();
tree_bkg->Scan();
tree_bkg->Write();
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("precoiltNF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("ytNF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qzpartOF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qrpart1OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("pzpartOF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("prpartOF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qsummaxOF"));



tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTNFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("ptNF"));
```

```
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTOFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTglitch1OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("PTlfnoise1OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("QS1OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qsum1OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("QS2OFchisq"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qsum2OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qo1OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qo2OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qi1OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qi2OF"));
tree_bkg->ResetBranchAddress(tree_bkg->GetBranch("qimean"));



tree_bkg->Scan();
fout->Close();

/*
Double_t ylinefun(double x)
{
return 0.95*x + 10 ; }
const double m= 0.95; //slope
const double c=-10; // y intecept

//TLine *line1 = new TLine(0,-10,170,60);
//TLine *line1 = new TLine((3-c)/m,3,80,ylinefun(80));
// line1->SetLineWidth(4);
// line1->SetLineColor(2);
//line1->Draw("same");
//TLine *line2 = new TLine(0,3,(3-c)/m,3); // y=3 line
//   line2->SetLineWidth(4);
//line2->SetLineColor(2);
//line2->Draw("same");
h_ERNRallcuts->Print();


qptNF->Update();

// Save the output plot in .gif form
qptNF->SaveAs(Form("qiprecoiltNF_RRQfill_%s_%s_%s.gif",
     fileaddress.c_str(),det.c_str(),weektitle.c_str()));
*/


}
```

8.4.3 WIMP model construction.

```
%#This is a MATLAB/OCTAVE code.
clear all;
load efficiency.txt;

[n,p]=size(efficiency);
eff=efficiency;

x_eff=eff(:,1);
y_eff=eff(:,2);

lerr_eff=eff(:,3); #lower error
uerr_eff=eff(:,4);

mass_dm=100;

load Cf_data.txt;
[n,p]=size(Cf_data)
A=Cf_data;


figure 1;
[hax, h1, h2] = plotyy(x_eff, ratefun(x_eff,mass_dm), x_eff, y_eff,'plot');

set(h1(1),'LineStyle','-','Marker','+','MarkerSize',4,'MarkerEdgeColor','b','Linewidth',2);
%set(h1(2),'LineStyle',' none','Marker','o',' MarkerSize',6,'MarkerEdgeColor','b');
set(h2(1),'LineStyle','-','Marker','+','MarkerSize',3,'MarkerEdgeColor','r','Linewidth',2);
title("WIMP Model",'FontSize',18,'FontWeight','bold');
xlabel(hax(1),"E_R ( keV)",'FontSize',18,'FontWeight','bold');
ylabel(hax(1), "WIMP spectrum",'FontSize',18,'FontWeight','bold');
ylabel(hax(2), "Detector eff.");

%# multiply theory rate X eff



mul= y_eff.*ratefun(x_eff,mass_dm);
hold on
plot(x_eff,mul,'k','Linewidth',2,'Markersize',7)

hold on;
%figure 2;

xCf=A(:,1);
yCf=0.75*power(10,-7)*A(:,2)/1.68355;
yerr=A(:,3);  # counts of Cf data

plot(xCf,yCf,'g','Linewidth',2,'Markersize',7);
```

```
l=legend(sprintf('WIMP spectrum m=%d GeV/c^2 ',mass_dm),
"Corrected spectrum","Cf data","Det eff")

set (1, "fontsize", 14)



#figure 2;
#plot(x_eff, y_eff ,'k','Linewidth',2,'Marker','+','Markersize',7)
#hold on;
#figure 3;
#plot(xCf2, det_eff_inter ,'r','Linewidth',2,'Marker','*','Markersize',2)

xCf2=A(10:425,1);
 det_eff_inter =interp1(x_eff, y_eff ,xCf2,'spline');
% #interpolated detector efficiency on points of xCf
yCf2=yCf(10:425,1);
mul_inter=interp1(x_eff,mul,xCf2,'spline');

figure 2; plot( x_eff ,mul,'r','Linewidth',2,'Marker','*','Markersize',2)
hold on;
plot(xCf2,mul_inter,'k','LineStyle','-','Linewidth',1,'Marker','+','Markersize',4)
l=legend(sprintf('Corrected spectrum for m=%d GeV/c^2',mass_dm),
"Interpolated corrected spectrum at Cf data points")
grid("on");
xlabel({'E_R ','( keV)'},'FontSize',18,'FontWeight','bold');
ylabel({'Rate','( units)'},'FontSize',18,'FontWeight','bold');
 title ("Corrected spectrum",'FontSize',18,'FontWeight','bold');
#limits= axis ([0 150])
xlim([0 150])
ylim([0 inf])

figure 3;

yw=mul_inter./yCf2;

plot(xCf2,yw,'o','Linewidth',2,'Markersize',5)

 title ("Weight vector distribution",'FontSize',18,'FontWeight','bold');

xlabel({'E_R ','( keV)'},'FontSize',18,'FontWeight','bold');
ylabel({'Weight values',''},'FontSize',18,'FontWeight','bold');
xlim([0 150])
ylim([0 inf])

//function definition :

function R=erfratefun(Er,mx) %Er is in keV ; 1 keV=1.609 X 10^(-16) Joules
% mx is Dark matter mass
% A=72.64 for Ge
mt=63.062597376; % Mass of target =0.932A in GeV/c^2
```

```
%Er=2*mt*5885.6226*power(10,-10)*mx*mx/(mx+mt)^2; %v0=230 km/s

v0=220; %most probable WIMP velocity in km/s
vE=232; % mean circular velocity of Earth through the halo
vesc=544; % Galactic escape velocity

uT=mt*mx/(mt+mx); % units of mass in GeV/c2 % 1 GeV/c2=1.79 X 10^(-27) kg ;
%1 keV/c2=1.79 X 10^(-33) kg
 scalefactor =sqrt(mt/2.0)*(2.998137224*10^2)*(1/uT);
vmin=Er.^(0.5);
vmin=scalefactor.*vmin;
[m,n]=size(vmin)
k0byk1=1/0.993361484;
R=zeros(m,n);
 for  i=1:n
x=vmin(1,i)/v0;
y=vE/v0;
z=vesc/v0;
 if  x>=0 && x<z-y
R(1,i)=erf(x+y)-erf(x-y)-(4/sqrt(pi))*y*exp(-z^2); endif
 if  x>z-y && x<z+y
R(1,i)=erf(z)-erf(x-y)-(2/sqrt(pi))*(y+z-x)*exp(-z^2); endif
 if  x>y+z
R(1,i)=0; endif
i
 fprintf ('Value of  vmin/v0 \n')
vmin(1,i)/v0
endfor




%R0=power(10,-8)*503/(mx*mt);
%y=(R0)*(1/Er)*exp(-x*power(10,-6)/Er);
%x is in  keV,but we need to put it  in  GeV scale, so  10^-6 factor in exponential

endfunction
```

8.4.4 Phonon radial FV cut.

```
#include "example_loadAnalysisFilesnewba.C" //Load Prodv5-6-3
#include "BasicCuts.h"                      //Load only Basic Cuts
#include <iostream>
#include <fstream>
#include <iomanip>
#include <TMath.h>
#include <iostream>
```

```
#include <fstream>

#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TFile.h>



void pradialcut_ba(string weekno="1", Int_t zip=4)
{
TGaxis::SetMaxDigits(3);
gStyle->SetLabelSize(0.025, "Z");

//Define Filename variable, Not used anywhere, if unnecesary then discard this line later
string filename, fileaddress ;
filename="ba";
fileaddress ="ba";

//Convert weekno from string to int for wherever it might be convenient
Int_t wno= atoi(Form("%s", weekno.c_str()));

//Define source for proper naming of output file
string source, sourcefullname;


source="Ba";
sourcefullname="Barium";
// Initialize zlite
Int_t zlite ;
if(zip==4)
{zlite=14;}
else
{
if(zip==14)
{zlite=4;}
else
{
if(zip==5 || weekno=="Allz4" )
{
if(wno>=1 && wno<7)
zlite=14; cout<<"zlite: "<< zlite<< endl;

}
else
{
if(zip==5 || weekno=="Allz14")
{
if(wno>6 && wno<=12)
zlite=4; cout<<"zlite: "<< zlite<< endl;
}
```

```
else
{
cout<<"Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1-12, or All or Allz4 or Allz14"<<endl;

}
}
}
}

// Initialize  weekno2
string weekno2=weekno;
if (weekno=="All" || weekno=="Allz4" || weekno=="Allz14")
{
weekno="*";
}

if (weekno2=="Allz4" && zip==4) //Added for convenience in running shell script
{
cout<<"Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1-12, or All or Allz4 or Allz14"<<endl;

}

if (weekno2=="Allz14" && zip==14) //Added for convenience in running shell script
{
cout<<"Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1-12, or All or Allz4 or Allz14"<<endl;

}

// Initialize  det
string det;
if (zip==4)
{det="T2Z1";}
else
{
if (zip==14)
{det="T5Z2";}
else
{
if (zip==5)
{det="T2Z2";}
else
{cout<<"Please enter either zip 4, 5 or 14";

}
}
}
// Initialize  weektitle, Not used in this  script  but maybe useful in others
string weektitle;
```

```
if (weekno2==”All” && det==”T2Z1”)
{
weektitle=”Weeks1−6”;
}
else
{

if (weekno2==”All” && det==”T5Z2”)
{
weektitle=”Weeks7−12”;
}
else
{
if (det==”T2Z2” && weekno2==”Allz4”)
{weektitle=”Weeks5−6”;}
else
{
if (det==”T2Z2” && weekno2==”Allz14”)
{weektitle=”Weeks7−12”;}
else
{
if (source==”Sb” && weekno2==”All”)
{//For Sb
weektitle=”Weeks1−9”;
}
else
{weektitle=”Week”;
weektitle=weektitle+weekno2;
}
}
}
}
}



//−−−−−−−−− Output File Address −−−−−−−−−−
//string opfileaddress=Form(”/home/viraj/Documents/DarkMatter/HT_Analysis/mf%s”,
    det.c_str());
string opfileaddress=”/home/u1/rik/Viraj/FV/phononradial”;

//Setting up detector indices
Int_t DetIndex = zip−1;
Int_t tid = (DetIndex/3)+1;
Int_t zid = (DetIndex%3)+1;
cout<<”\nWe are looking at: T”<<tid<<”Z”<<zid<<endl;

//Confirming BiasFlashtime
//cout<<”BiasFlashTime= ”<<flashtime()<<endl;

// −−−−−−−−−−−− Location of Data −−−−−−−−−−−−
```

```
// --- Loading Prodv5-6-3 data ---
//string dataDir = "/home/viraj/Documents/DarkMatter/HT_Analysis/";
string dataDir = "/data/R134/dataReleases/Prodv5-3-5/merged/all/";

string fileaddress2;
fileaddress2 =fileaddress;
dataDir = dataDir+fileaddress2;
string seriesweek="week";
seriesweek=seriesweek+weekno;

//Call function that loads data
TChain *chain2 = chainDataAllSpecial(zip, dataDir, weekno, fileaddress, zlite);
//chain2->Scan();
cout<<"Number of events: "<<chain2->GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2->GetName()<<endl<<endl;


//Define some Quality cuts
TCut cLEChisq("PTNFchisq < (0.015*ptNF*ptNF+4500) && PTNFchisq >3600");
TCut cDeltaChisqGlitch("(PTOFchisq-PTglitch1OFchisq)<((-2.3*(ptNF*ptNF))+25)");
TCut cDeltaChiSqLFNLine("(PTOFchisq-PTlfnoise1OFchisq)< 14.2");
TCut cDeltaChiSqLFNParabola("(PTOFchisq-PTlfnoise1OFchisq)<
((-13.5152*(ptNF*ptNF))-(2.18824*ptNF)+27.692)");
TCut cDeltaChiSqLFN=cDeltaChiSqLFNLine+cDeltaChiSqLFNParabola;
TCut cChargeChiSqS1("QS1OFchisq <= ((( 0.00578106)*qsum1OF*qsum1OF*qsum1OF)
-( 0.427417*qsum1OF*qsum1OF)+( 9.88895*qsum1OF)+ 5448.68)");
TCut cChargeChiSqS2("QS2OFchisq <= ((( 0.00327275)*qsum2OF*qsum2OF*qsum2OF)
-( 0.25359*qsum2OF*qsum2OF)+( 7.05146*qsum2OF)+ 6241.38)");
TCut cChargeChiSq=cChargeChiSqS1+cChargeChiSqS2;

TCut cRadialCutS1("qo1OF<3.0 || qi1OF>3.0");
TCut cRadialCutS2("qo2OF<1.8 || qi2OF>3.0");
TCut cRadialCut=cRadialCutS1+cRadialCutS2;

TCut cFVPolyUp("qi2OF<(-0.00233946*qi1OF*qi1OF)+1.11534*qi1OF+2.06405");
TCut cFVPolyLow("qi2OF>(-0.0035754*qi1OF*qi1OF)+1.04275*qi1OF-2.85197");
TCut cFVPoly=cFVPolyUp+cFVPolyLow;
TCut cHorizontalLine("qi2OF<3.5"); //mu+5.5*sigma of qi2OF
TCut cVerticalLine("qi1OF<3.5"); // mu+13*sigma of qi1OF
TCut cFlatLines=cHorizontalLine+cVerticalLine;
TCut cSymmetricFV=(cFVPoly)||(cFlatLines);

TCut cZeroCharge("((qi1OF+qi2OF)/2)<1 && ((qi1OF+qi2OF)/2)>-1");
TCut cPhononRadial("prpartOF<(TMath::Exp(-2.42429-0.0990673*ptNF)+0.241877)"); // 2
    sigma
TCut cNotPhononRadial("prpartOF>(TMath::Exp(-2.42429-0.0990673*ptNF)+0.241877)"); //
    2 sigma

//TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > (70/170)*ptNF -10"); // simulation for 3
    sigma NR band cut
```

```
TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > 0.95*precoiltNF -10 && ((qi1OF+qi2OF)/2.0)
    >3 "); // simulation for 3 sigma NR band cut with ptNF on x axis

TCut
    cQualityCuts=cLEChisq+cDeltaChisqGlitch+cDeltaChiSqLFN+cChargeChiSq+cSymmetricFV;

TCut virajPhononRadial("prpartOF<(TMath::Exp(-0.424675-3.53768*ptNF)+0.295494)"); //
    2 sigma
TCut virajchargeradialcut("qo1OF<1.05087 && qo2OF<1.23711");

cout<<"\nNumber of events: "<<chain2->GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2->GetName()<<endl;

//chain2->Scan("SeriesNumber","SeriesNumber==11509070319", "colsize=15 precision=13");


double ptNFbinsize=150;
double ptNFrange_1a=0;
double ptNFrange_2a=15;

double prpartOF_bins=200;
double prpartOF_range1=0.1;
double prpartOF_range2=0.5;

gStyle->SetOptFit(1);


TH2D *h_phonon = new TH2D("h_phonon", Form("Z%d: Barium-%s,prpart vs ptNF",zid,
    filename.c_str()), ptNFbinsize, ptNFrange_1a, ptNFrange_2a,
prpartOF_bins, prpartOF_range1, prpartOF_range2);
h_phonon->SetXTitle("ptNF [keV]");
h_phonon->SetYTitle("prpartOF");
h_phonon->GetYaxis()->SetTitleOffset(1.5);
h_phonon->GetYaxis()->SetLabelSize(0.03);
h_phonon->GetYaxis()->SetTitleSize(0.03);
h_phonon->GetYaxis()->CenterTitle(true);
h_phonon->GetXaxis()->CenterTitle(true);
h_phonon->SetMarkerColor(kRed);
h_phonon->SetFillColor(kRed);
//h_phonon->SetStats(kFALSE);

TCanvas* canvas_th2dphonon= new TCanvas("canvas_th2dphonon", "Prpart Vs ptNF canvas",
    600, 600);
chain2->Draw("prpartOF:ptNF>>h_phonon",cBasicCuts);

cout<<" Bin 1st: "<<h_phonon->GetXaxis()->FindBin(0)<<" Bin last: "<<
    h_phonon->GetXaxis()->FindBin(1)<<endl;
//Projection of Y
TF1 *gausfit=new TF1("gausfit","gaus", prpartOF_range1, prpartOF_range2);

TF1 *polyfit=new TF1("polyfit","TMath::Exp(-[0]-[1]*x)+[2]", ptNFrange_1a, ptNFrange_2a);
```

```
polyfit->SetLineColor(1); // 3 sigma
polyfit->SetParameters(-0.8,5,0.2); // For 3 and 2.5 sigma fits



double binw=(ptNFrange_2a-ptNFrange_1a)/ptNFbinsize;
//cout<<endl<<"binw of ptNF is: "<<binw<<endl;
//TCanvas* CanprojY= new TCanvas("CanprojY", "Projection Y", 600, 600);

/*
TH1D *hist_projY;
//hist_projY=h_phonon->ProjectionY("hist_projY", h_phonon->GetXaxis()->FindBin(0),
    h_phonon->GetXaxis()->FindBin(1));
hist_projY=h_phonon->ProjectionY("hist_projY", h_phonon->GetXaxis()->FindBin(1),
    h_phonon->GetXaxis()->FindBin(1));
hist_projY->SetYTitle("counts");
hist_projY->GetYaxis()->SetTitleOffset(1.5);
hist_projY->Rebin(5);
hist_projY->Fit("gausfit");
TF1 *f = hist_projY->GetListOfFunctions()->FindObject("gausfit");
if(f){
f->SetLineColor(3); //3=green 4 = "blue"
f->SetLineWidth(4.5); //f->SetLineStyle(1); // 2 = "- - -"
}


//hist_projY->SetAxisRange(0.1, 0.2,"X");
hist_projY->Draw();
*/
//Define histogram for mean of psumo vs ptnf side 1
TH2D *h_psumptnfmeanS1 = new TH2D("h_psumptnfmeanS1", Form("T%dZ%d: Barium, %s,
    prpartOF vs ptNF", tid, zid, weektitle.c_str()), ptNFbinsize, ptNFrange_1a, ptNFrange_2a,
    prpartOF_bins, prpartOF_range1, prpartOF_range2);
h_psumptnfmeanS1->SetXTitle("ptNF [keV]");
h_psumptnfmeanS1->SetYTitle("prpartOF");
h_psumptnfmeanS1->GetYaxis()->SetTitleOffset(1.5);
h_psumptnfmeanS1->GetYaxis()->SetLabelSize(0.03);
h_psumptnfmeanS1->GetYaxis()->SetTitleSize(0.03);
h_psumptnfmeanS1->SetMarkerColor(5);
h_psumptnfmeanS1->SetMarkerSize(1);
h_psumptnfmeanS1->SetMarkerStyle(20);
//h_psumptnfmeanS1->SetStats(kFALSE);

//Define histogram for 1 sigma of psumo vs ptnf side 1
TH2D *h_psumptnfsigmaS1 = new TH2D("h_psumptnfsigmaS1", Form("T%dZ%d: Barium, %s,
    prpartOF vs ptNF", tid, zid, weektitle.c_str()), ptNFbinsize, ptNFrange_1a, ptNFrange_2a,
    prpartOF_bins, prpartOF_range1, prpartOF_range2);
h_psumptnfsigmaS1->SetXTitle("ptNF [keV]");
h_psumptnfsigmaS1->SetYTitle("prpartOF");
h_psumptnfsigmaS1->GetYaxis()->SetTitleOffset(1.5);
h_psumptnfsigmaS1->GetYaxis()->SetLabelSize(0.03);
```

```
h_psumptnfsigmaS1->GetYaxis()->SetTitleSize(0.03);
h_psumptnfsigmaS1->SetMarkerColor(4);
h_psumptnfsigmaS1->SetMarkerSize(3);
h_psumptnfsigmaS1->SetMarkerStyle(3);
//h_psumptnfsigmaS1->SetStats(kFALSE);

const int nbins=20; //Previously 222

TH1D *py[nbins];
TCanvas* poptS1[nbins];
TLatex *t3[nbins];


double var1_ptNF, var2_ptNF;
for(int i=0; i<nbins; i++)
{cout<<"------------------- Bin No:
    "<<i<<"--------------------"<<endl;
poptS1[i]= new TCanvas(Form("poptS1_%d", i), Form("prpartOF bin%d",i), 600, 600);
py[i]=h_phonon->ProjectionY(Form("py%d", i), i, i);
var1_ptNF=i*binw; var2_ptNF=var1_ptNF+binw; cout<<"var1_ptNF =
    "<<var1_ptNF<<"var2_ptNF = "<<var2_ptNF<<endl;
py[i]->Rebin(5);
py[i]->SetTitle(Form("T%dZ%d: Barium, %s, prpartOF, Bin No. %d, %.2f <ptNF < %.2f",
    tid, zid, weektitle.c_str(), i,var1_ptNF, var2_ptNF ));
py[i]->SetYTitle("counts");
py[i]->GetYaxis()->SetTitleOffset(1.5);
py[i]->GetYaxis()->SetLabelSize(0.03);
py[i]->GetYaxis()->SetTitleSize(0.03);
//py[i]->SetStats(kFALSE);
poptS1[i]->cd();
py[i]->Draw();

py[i]->Fit("gausfit");
TF1 *f = py[i]->GetListOfFunctions()->FindObject("gausfit");
if(f){
f->SetLineColor(3); //3=green 4 = "blue"
f->SetLineWidth(4.5); //f->SetLineStyle(1); // 2 = "- - -"
}
//t3[i]=new TLatex(.55,.8,Form("#scale[0.7]{%.2f #leq ptNF #leq %.2f}",
    (nbins-(nbins-(i-1)))*binw, i*binw));
//t3[i]->SetNDC(kTRUE);
//t3[i]->Draw("same");
//poptS1[i]->SaveAs(Form("%s/New/Bins/prptbin%d_%s_%s_%s.gif", opfileaddress.c_str(), i,
    fileaddress.c_str(),det.c_str(), weektitle.c_str()));

h_psumptnfmeanS1->Fill(((i*binw)-(binw/2)), gausfit->GetParameter(1));
h_psumptnfsigmaS1->Fill(((i*binw)-(binw/2)),
    (gausfit->GetParameter(1)+2*gausfit->GetParameter(2)));
//h_psumptnf3sigmaS1->Fill(((i*binw)-(binw/2)-10),
    (gausfit->GetParameter(1)+(2*gausfit->GetParameter(2))));
```

```
}

TCanvas* canvas_mean_sigma= new TCanvas("canvas_mean_sigma", "Side 1 All Means and
    sigmas histogram", 600, 600);
h_phonon->Draw();
//h_psumptnfmeanS1->Draw("same");
h_psumptnfsigmaS1->Draw("same");
h_psumptnfsigmaS1->Fit("polyfit", "R");

TLegend *leg = new TLegend(0.35,0.7,0.45,0.8,NULL,"brNDC");
leg->SetBorderSize(0);
leg->SetTextSize(0.03);
leg->SetLineColor(1);
leg->SetLineStyle(1);
leg->SetLineWidth(1);
leg->SetFillColor(0);
leg->SetFillStyle(0);
leg->AddEntry("h_phonon","Data","lpf");
leg->AddEntry("h_psumptnfsigmaS1","mean+sigma of ProjectionY","lpf");
leg->AddEntry("polyfit","Exponential Fit","lpf");
leg->Draw("same");

// TCanvas* canvas_aftercut= new TCanvas("canvas_after cut", "After phonon radial cut", 600,
    600);
//chain2->Draw("prpartOF:ptNF",cBasicCuts && virajPhononRadial && "ptNF<15" &&
    "ptNF>0" && "prpartOF>0.1" && "prpartOF<0.5");
chain2->Draw("0.5*(qi1OF+qi2OF):precoiltNF",cBasicCuts && virajPhononRadial &&
    "qi1OF+qi2OF>0" && "0.5*(qi1OF+qi2OF)<160 " && "precoiltNF>0" &&
    "precoiltNF<160");


}
```

8.4.5 Charge Radial FV cut.

```
#include "example_loadAnalysisFilesnewba.C" //Load Prodv5-6-3
#include "BasicCuts.h"                      //Load only Basic Cuts
#include <iostream>
#include <fstream>
#include <iomanip>
#include <TMath.h>
#include <iostream>
#include <fstream>

#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
```

```
#include <TFile.h>

Double_t Normal(Double_t *x, Double_t *par)
{
//return (par[0]*par[1]*0.01*1)/(2*3.14159)/((x[0]-par[2])*(x[0]-par[2])+(par[1]*par[1])/4.);
Double_t y;
y=par[0]*exp( -0.5*pow( (x-par[1])/par[2] ,2 ) )          ;
return y;
}



void cradialcut_ba(string weekno="1", Int_t zip=4)
{
TGaxis::SetMaxDigits(3);
gStyle->SetLabelSize(0.025, "Z");

//Define Filename variable, Not used anywhere, if unnecesary then discard this line later
string filename, fileaddress ;
filename="ba";
fileaddress ="ba";

//Convert weekno from string to int for wherever it might be convenient
Int_t wno= atoi(Form("%s", weekno.c_str()));

//Define source for proper naming of output file
string source, sourcefullname;


source="Ba";
sourcefullname="Barium";
// Initialize zlite
Int_t zlite ;
if(zip==4)
{zlite=14;}
else
{
if(zip==14)
{zlite=4;}
else
{
if(zip==5 || weekno=="Allz4" )
{
if(wno>=1 && wno<7)
zlite=14; cout<<"zlite: "<< zlite<< endl;

}
else
{
if(zip==5 || weekno=="Allz14")
{
```

```
if (wno>6 && wno<=12)
 zlite =4; cout<<"zlite: "<< zlite<< endl;
}
else
{
cout<<"Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
      between 1-12, or All or Allz4 or Allz14"<<endl;

}
}
}
}

// Initialize  weekno2
string  weekno2=weekno;
if (weekno=="All" || weekno=="Allz4" || weekno=="Allz14")
{
weekno="*";
}

if (weekno2=="Allz4" && zip==4) //Added for convenience in running shell script
{
cout<<"Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
      between 1-12, or All or Allz4 or Allz14"<<endl;

}

if (weekno2=="Allz14" && zip==14) //Added for convenience in running shell script
{
cout<<"Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
      between 1-12, or All or Allz4 or Allz14"<<endl;

}

// Initialize  det
string  det;
if (zip==4)
{det="T2Z1";}
else
{
if (zip==14)
{det="T5Z2";}
else
{
if (zip==5)
{det="T2Z2";}
else
{cout<<"Please enter either zip 4, 5 or 14";

}
}
```

```
}
// Initialize  weektitle, Not used in this  script  but maybe useful in others
string  weektitle;
if(weekno2==”All” && det==”T2Z1”)
{
weektitle=”Weeks1-6”;
}
else
{

if(weekno2==”All” && det==”T5Z2”)
{
weektitle=”Weeks7-12”;
}
else
{
if(det==”T2Z2” && weekno2==”Allz4”)
{weektitle=”Weeks5-6”;}
else
{
if(det==”T2Z2” && weekno2==”Allz14”)
{weektitle=”Weeks7-12”;}
else
{
if(source==”Sb” && weekno2==”All”)
{//For Sb
weektitle=”Weeks1-9”;
}
else
{weektitle=”Week”;
weektitle=weektitle+weekno2;
}
}
}
}
}



//--------- Output File Address ------------
//string  opfileaddress=Form(”/home/viraj/Documents/DarkMatter/HT_Analysis/mf%s”,
    det.c_str());
string  opfileaddress=”/home/u1/rik/Viraj/FV/chargeradialcut”;

//Setting up detector  indices
Int_t  DetIndex = zip-1;
Int_t  tid  = (DetIndex/3)+1;
Int_t  zid  = (DetIndex%3)+1;
cout<<”\nWe are looking at: T”<<tid<<”Z”<<zid<<endl;

//Confirming BiasFlashtime
```

```
//cout<<"BiasFlashTime= "<<flashtime()<<endl;

// ---------- Location of Data ----------
// --- Loading Prodv5-6-3 data ---
//string dataDir = "/home/viraj/Documents/DarkMatter/HT_Analysis/";
string dataDir = "/data/R134/dataReleases/Prodv5-3-5/merged/all/";

string fileaddress2;
fileaddress2 =fileaddress;
dataDir = dataDir+fileaddress2;
string seriesweek="week";
seriesweek=seriesweek+weekno;

//Call function that loads data
TChain *chain2 = chainDataAllSpecial(zip, dataDir, weekno, fileaddress, zlite);
//chain2->Scan();
cout<<"Number of events: "<<chain2->GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2->GetName()<<endl<<endl;

//Define Variables
double ptNFrange_1=-10; // Prev: -10, 0
double ptNFrange_2=170; // Prev: 170, 30
double ptNFbinsize=450; // Prev: 450, 300
double qbinsize=500; // Prev: 500, 160
double q_y_range_1=-10; //Prev: -10, -2
double q_y_range_2= 80; //Prev: 80, 14

const double binw= (ptNFrange_2-ptNFrange_1)/ptNFbinsize;


//Define some Quality cuts
TCut cLEChisq("PTNFchisq < (0.015*ptNF*ptNF+4500) && PTNFchisq >3600");
TCut cDeltaChisqGlitch("(PTOFchisq-PTglitch1OFchisq)<((-2.3*(ptNF*ptNF))+25)");
TCut cDeltaChiSqLFNLine("(PTOFchisq-PTlfnoise1OFchisq)< 14.2");
TCut cDeltaChiSqLFNParabola("(PTOFchisq-PTlfnoise1OFchisq)
<((-13.5152*(ptNF*ptNF))-(2.18824*ptNF)+27.692)");
TCut cDeltaChiSqLFN=cDeltaChiSqLFNLine+cDeltaChiSqLFNParabola;
TCut cChargeChiSqS1("QS1OFchisq <= ((( 0.00578106)*qsum1OF*qsum1OF*qsum1OF)
-( 0.427417*qsum1OF*qsum1OF)+( 9.88895*qsum1OF)+ 5448.68)");
TCut cChargeChiSqS2("QS2OFchisq <= ((( 0.00327275)*qsum2OF*qsum2OF*qsum2OF)
-( 0.25359*qsum2OF*qsum2OF)+( 7.05146*qsum2OF)+ 6241.38)");
TCut cChargeChiSq=cChargeChiSqS1+cChargeChiSqS2;

TCut cRadialCutS1("qo1OF<3.0 || qi1OF>3.0");
TCut cRadialCutS2("qo2OF<1.8 || qi2OF>3.0");
TCut cRadialCut=cRadialCutS1+cRadialCutS2;

TCut cFVPolyUp("qi2OF<(-0.00233946*qi1OF*qi1OF)+1.11534*qi1OF+2.06405");
TCut cFVPolyLow("qi2OF>(-0.0035754*qi1OF*qi1OF)+1.04275*qi1OF-2.85197");
TCut cFVPoly=cFVPolyUp+cFVPolyLow;
TCut cHorizontalLine("qi2OF<3.5"); //mu+5.5*sigma of qi2OF
```

```
TCut cVerticalLine("qi1OF<3.5"); // mu+13*sigma of qi1OF
TCut cFlatLines=cHorizontalLine+cVerticalLine;
TCut cSymmetricFV=(cFVPoly)||(cFlatLines);

TCut cZeroCharge("((qi1OF+qi2OF)/2)<1 && ((qi1OF+qi2OF)/2)>-1");
TCut cPhononRadial("prpartOF<(TMath::Exp(-2.42429-0.0990673*ptNF)+0.241877)"); // 2
    sigma
TCut cNotPhononRadial("prpartOF>(TMath::Exp(-2.42429-0.0990673*ptNF)+0.241877)"); //
    2 sigma

//TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > (70/170)*ptNF -10"); // simulation for 3
    sigma NR band cut
TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > 0.95*precoiltNF -10 && ((qi1OF+qi2OF)/2.0)
    >3 "); // simulation for 3 sigma NR band cut with ptNF on x axis

TCut
    cQualityCuts=cLEChisq+cDeltaChisqGlitch+cDeltaChiSqLFN+cChargeChiSq+cSymmetricFV;

TCut virajPhononRadial("prpartOF<(TMath::Exp(-0.424675-3.53768*ptNF)+0.295494)"); //
    2 sigma
TCut virajchargeradialcut("qo1OF<1.05087 && qo2OF<1.23711");


cout<<"\nNumber of events: "<<chain2->GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2->GetName()<<endl;

//chain2->Scan("SeriesNumber","SeriesNumber==11509070319", "colsize=15 precision=13");

double qi1OF_range_1=-2;
double qi1OF_range_2=10;
double qi2OF_range_1=-4;
double qi2OF_range_2=10;
double qo1OF_range_1=-2;
double qo1OF_range_2=5;
double qo2OF_range_1=-2;
double qo2OF_range_2=10;
double qsum1OF_range_1=-4;
double qsum1OF_range_2=15;
double qsum2OF_range_1=-5;
double qsum2OF_range_2=15;
double qi1OF_bins=100;
double qo1OF_bins=100;
double qi2OF_bins=100;
double qo2OF_bins=100;
double qsum1OF_bins=100;
double qsum2OF_bins=100;

TH2D *h_charge = new TH2D("h_charge", Form("Z%d: Barium-%s, Side 1: Outer Charge vs
    Inner Charge",zid, filename.c_str()), qi1OF_bins, qi1OF_range_1, qi1OF_range_2, qo1OF_bins,
    qo1OF_range_1, qo1OF_range_2);
h_charge->SetXTitle("Inner Charge [keV]");
```

```
h_charge->SetYTitle("Outer Charge [keV]");
h_charge->GetYaxis()->SetTitleOffset(1.5);
h_charge->GetYaxis()->SetLabelSize(0.03);
h_charge->GetYaxis()->SetTitleSize(0.03);
h_charge->SetMarkerColor(kRed);
h_charge->SetFillColor(kRed);
h_charge->SetStats(kFALSE);

TCanvas *ch2d= new TCanvas("ch2d", "cTH2D outer charge vs inner charge", 600, 600);
chain2->Draw("qo1OF:qi1OF>>h_charge",cBasicCuts);
//chain2->Draw("qo1OF:qi1OF>>h_charge");
//chain2->Draw("qo1OF:qi1OF>>h_charge",
    cNotEmpty+cRandom+cGoodFlash+cPstd+cBaseTemp+cVoltageBias, "same");
//ch2d->SaveAs(Form("%s/OuterCharge vs Innercharge_side1_%s_%s_%s.gif",
    opfileaddress.c_str(), fileaddress.c_str(),det.c_str(), weektitle.c_str()));



TH1D *S1proj_inb, *S1proj_onb, *S1proj_inevents;

TCanvas* CanQP2= new TCanvas("CanQP2", "S1: Outer Noise Events Projection", 600, 600);
S1proj_onb=h_charge->ProjectionY("S1proj_inevents", h_charge->GetXaxis()->FindBin(-2),
    h_charge->GetXaxis()->FindBin(2));
S1proj_onb->SetTitle(Form("Z%d: Barium-%s,%s, Side 1 Outer Noise Spectrum
    Projection",zid, filename.c_str(), weektitle.c_str()));
S1proj_onb->SetYTitle("counts");
S1proj_onb->GetYaxis()->SetTitleOffset(1.5);
S1proj_onb->GetYaxis()->SetLabelSize(0.03);
S1proj_onb->GetYaxis()->SetTitleSize(0.03);

//S1proj_onb->GetXaxis()->SetRange(-1,1);
//S1proj_onb->SetAxisRange(-0.8, 1,"X");

//S1proj_onb->SetStats(kFALSE);
S1proj_onb->Draw();
cout<<" Bin 0: "<<h_charge->GetXaxis()->FindBin(-2)<<" Bin 1: "<<
    h_charge->GetYaxis()->FindBin(2)<<endl;

/*
double fitlow=-1.0;
double fithi=1.0;
TF1 *fitFunNormal = new TF1("fitFunNormal",Normal,fitlow,fithi,3);
fitFunNormal->SetParameter(0,250);
fitFunNormal->SetParameter(1,0.5); // mean set
fitFunNormal->SetParameter(2,0.5); // sigma set
//fitFunNormal->FixParameter(1,0.048);

//fitFunNormal->SetParLimits(0,100,1000);
//fitFunNormal->SetParLimits(1,-1,1);
//fitFunNormal->SetParLimits(2,0,1);
//fitFunNormal->SetLineColor(3);
```

```
fitFunNormal->SetParNames("Amplitude","Mean","Sigma");

S1proj_onb->Fit(fitFunNormal,"IER+");

float  A=fitFunNormal->GetParameter(0);
float  mu=fitFunNormal->GetParameter(1);
float  sigma=fitFunNormal->GetParameter(2);

*/

S1proj_onb->Fit("gaus");
gaus->SetLineColor(3);

float  A=gaus->GetParameter(0);
float  mu=gaus->GetParameter(1);
float  sigma=gaus->GetParameter(2);

cout<<" mu "<< mu<<" 3 X sigma= "<<3*sigma<<endl;
cout<<" mu "<< mu<<"mu + 3sigma= "<<mu+3*sigma<<endl;

TLegend *leg = new TLegend(0.2,0.7,0.3,0.8,NULL,"brNDC");
leg->SetBorderSize(0);
leg->SetTextSize(0.03);
leg->SetLineColor(1);
leg->SetLineStyle(1);
leg->SetLineWidth(1);
leg->SetFillColor(0);
leg->SetFillStyle(0);
leg->AddEntry("S1proj_onb","Projection Outer Charge ","lpf");
leg->AddEntry("gaus","Gaussian Fit","lpf");

leg->Draw("same");
//cout<<"Outer Noise Mean: "<<S1proj_onb->GetMean()<<endl;
//cout<<"Outer Noise StdDev: "<<S1proj_onb->GetStdDev()<<endl;
//cout<<"Outer Noise Mean+5*StdDev(D)=
    "<<S1proj_onb->GetMean()+5*S1proj_onb->GetStdDev()<<endl;
//CanQP2->SaveAs(Form("%s/canvas_S1OuterNoiseProj_%s_%s_%s.C", opfileaddress.c_str(),
    fileaddress.c_str(),det.c_str(),weektitle.c_str()));


TCanvas *ch2dcut_side1= new TCanvas("ch2dcut_side1", "cTH2D outer charge vs inner charge",
    600, 600);
ch2dcut_side1->cd();
chain2->Draw("qo1OF:qi1OF>>h_charge",cBasicCuts);
TLine *line1 = new TLine(-2,mu+3*sigma,10,mu+3*sigma);
line1->SetLineWidth(4);
line1->SetLineColor(3);
line1->Draw("same");
TLegend *legcut = new TLegend(0.5365772,0.7801394,0.6708054,0.8797909,NULL,"brNDC");
legcut->SetBorderSize(0);
legcut->SetTextSize(0.03);
```

```
legcut->SetLineColor(1);
legcut->SetLineStyle(1);
legcut->SetLineWidth(1);
legcut->SetFillColor(0);
legcut->SetFillStyle(0);
TLegendEntry *entryc=legcut->AddEntry("line1","Charge radial cut ","p");
legcut->Draw("same");
entryc->SetTextFont(42);
//chain2->Draw("qo1OF:qi1OF>>h_charge");
//chain2->Draw("qo1OF:qi1OF>>h_charge",
    cNotEmpty+cRandom+cGoodFlash+cPstd+cBaseTemp+cVoltageBias, "same");
//ch2dcut->SaveAs(Form("%s/canvas_chargeradialcut_OuterChargevsInnercharge_side1_%s_%s_%s.C",
    opfileaddress.c_str(), fileaddress.c_str(),det.c_str(),weektitle.c_str()));
//ch2dcut->SaveAs(Form("%s/canvas_cut_OuterCharge vs Innercharge_side1_%s_%s_%s.gif",
    opfileaddress.c_str(), fileaddress.c_str(),det.c_str(), weektitle.c_str()));


}
```

8.4.6 Charge Symmetric FV cut.

```
#include "example_loadAnalysisFilesnewba.C" //Load Prodv5-6-3
#include "BasicCuts.h"                      //Load only Basic Cuts
#include <iostream>
#include <fstream>
#include <iomanip>
#include <TMath.h>
#include <iostream>
#include <fstream>

#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>
#include <TFile.h>



void symcut_ba(string weekno="1", Int_t zip=4)
{
TGaxis::SetMaxDigits(3);
gStyle->SetLabelSize(0.025, "Z");

//Define Filename variable, Not used anywhere, if unnecesary then discard this line later
string filename, fileaddress ;
```

```
filename="ba";
 fileaddress ="ba";

//Convert weekno from string to int for wherever it might be convenient
Int_t wno= atoi(Form("%s", weekno.c_str()));

//Define source for proper naming of output file
string source, sourcefullname;


source="Ba";
sourcefullname="Barium";
// Initialize zlite
Int_t zlite ;
if (zip==4)
{ zlite =14;}
else
{
if (zip==14)
{ zlite =4;}
else
{
if (zip==5 || weekno=="Allz4" )
{
if (wno>=1 && wno<7)
zlite =14; cout<<"zlite: "<< zlite<< endl;

}
else
{
if (zip==5 || weekno=="Allz14")
{
if (wno>6 && wno<=12)
zlite =4; cout<<"zlite: "<< zlite<< endl;
}
else
{
cout<<"Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1-12, or All or Allz4 or Allz14"<<endl;

}
}
}
}

// Initialize weekno2
string weekno2=weekno;
if (weekno=="All" || weekno=="Allz4" || weekno=="Allz14")
{
weekno="*";
}
```

```
if (weekno2==”Allz4” && zip==4) //Added for convenience in running shell script
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}

if (weekno2==”Allz14” && zip==14) //Added for convenience in running shell script
{
cout<<”Incorrect zip or week number. Please select zip4, zip5 or zip14 and week numbers
    between 1−12, or All or Allz4 or Allz14”<<endl;

}

// Initialize det
string det;
if (zip==4)
{det=”T2Z1”;}
else
{
if (zip==14)
{det=”T5Z2”;}
else
{
if (zip==5)
{det=”T2Z2”;}
else
{cout<<”Please enter either zip 4, 5 or 14”;

}
}
}
// Initialize weektitle, Not used in this script but maybe useful in others
string weektitle;
if (weekno2==”All” && det==”T2Z1”)
{
weektitle=”Weeks1−6”;
}
else
{

if (weekno2==”All” && det==”T5Z2”)
{
weektitle=”Weeks7−12”;
}
else
{
if (det==”T2Z2” && weekno2==”Allz4”)
{weektitle=”Weeks5−6”;}
else
```

```
{
if (det==”T2Z2” && weekno2==”Allz14”)
{weektitle=”Weeks7-12”;}
else
{
if (source==”Sb” && weekno2==”All”)
{//For Sb
weektitle=”Weeks1-9”;
}
else
{weektitle=”Week”;
weektitle=weektitle+weekno2;
}
}
}
}
}

//---------- Output File Address ----------
//string opfileaddress=Form(”/home/viraj/Documents/DarkMatter/HT_Analysis/mf%s”,
    det.c_str());
string opfileaddress=”/home/u1/rik/Viraj/FV/symmetrycut”;

//Setting up detector indices
Int_t DetIndex = zip-1;
Int_t tid = (DetIndex/3)+1;
Int_t zid = (DetIndex%3)+1;
cout<<”\nWe are looking at: T”<<tid<<”Z”<<zid<<endl;

//Confirming BiasFlashtime
//cout<<”BiasFlashTime= ”<<flashtime()<<endl;

// -------- Location of Data ----------
// --- Loading Prodv5-6-3 data ---
//string dataDir = ”/home/viraj/Documents/DarkMatter/HT_Analysis/”;
string dataDir = ”/data/R134/dataReleases/Prodv5-3-5/merged/all/”;

string fileaddress2;
fileaddress2=fileaddress;
dataDir = dataDir+fileaddress2;
string seriesweek=”week”;
seriesweek=seriesweek+weekno;

//Call function that loads data
TChain *chain2 = chainDataAllSpecial(zip, dataDir, weekno, fileaddress, zlite);
//chain2->Scan();
cout<<”Number of events: ”<<chain2->GetEntries()<<endl;
cout<<”Name of Tree: ”<<chain2->GetName()<<endl<<endl;
```

```
//Define Variables
double ptNFrange_1=−10; // Prev: −10, 0
double ptNFrange_2=170; // Prev: 170, 30
double ptNFbinsize=450; // Prev: 450, 300
double qbinsize=500; // Prev: 500, 160
double q_y_range_1=−10; //Prev: −10, −2
double q_y_range_2= 80; //Prev: 80, 14

const double binw= (ptNFrange_2−ptNFrange_1)/ptNFbinsize;


//Define some Quality cuts
TCut cLEChisq("PTNFchisq < (0.015*ptNF*ptNF+4500) && PTNFchisq >3600");
TCut cDeltaChisqGlitch("(PTOFchisq−PTglitch1OFchisq)<((−2.3*(ptNF*ptNF))+25)");
TCut cDeltaChiSqLFNLine("(PTOFchisq−PTlfnoise1OFchisq)< 14.2");
TCut cDeltaChiSqLFNParabola("(PTOFchisq−PTlfnoise1OFchisq)<
((−13.5152*(ptNF*ptNF))−(2.18824*ptNF)+27.692)");
TCut cDeltaChiSqLFN=cDeltaChiSqLFNLine+cDeltaChiSqLFNParabola;
TCut cChargeChiSqS1("QS1OFchisq <= ((( 0.00578106)*qsum1OF*qsum1OF*qsum1OF)
−( 0.427417*qsum1OF*qsum1OF)+( 9.88895*qsum1OF)+ 5448.68)");
TCut cChargeChiSqS2("QS2OFchisq <= ((( 0.00327275)*qsum2OF*qsum2OF*qsum2OF)
−( 0.25359*qsum2OF*qsum2OF)+( 7.05146*qsum2OF)+ 6241.38)");
TCut cChargeChiSq=cChargeChiSqS1+cChargeChiSqS2;

TCut cRadialCutS1("qo1OF<3.0 || qi1OF>3.0");
TCut cRadialCutS2("qo2OF<1.8 || qi2OF>3.0");
TCut cRadialCut=cRadialCutS1+cRadialCutS2;

TCut cFVPolyUp("qi2OF<(−0.00233946*qi1OF*qi1OF)+1.11534*qi1OF+2.06405");
TCut cFVPolyLow("qi2OF>(−0.0035754*qi1OF*qi1OF)+1.04275*qi1OF−2.85197");
TCut cFVPoly=cFVPolyUp+cFVPolyLow;
TCut cHorizontalLine("qi2OF<3.5"); //mu+5.5*sigma of qi2OF
TCut cVerticalLine("qi1OF<3.5"); // mu+13*sigma of qi1OF
TCut cFlatLines=cHorizontalLine+cVerticalLine;
TCut cSymmetricFV=(cFVPoly)||(cFlatLines);

TCut cZeroCharge("((qi1OF+qi2OF)/2)<1 && ((qi1OF+qi2OF)/2)>−1");
TCut cPhononRadial("prpartOF<(TMath::Exp(−2.42429−0.0990673*ptNF)+0.241877)"); // 2
    sigma
TCut cNotPhononRadial("prpartOF>(TMath::Exp(−2.42429−0.0990673*ptNF)+0.241877)"); //
    2 sigma

//TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > (70/170)*ptNF −10"); // simulation for 3
    sigma NR band cut
TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > 0.95*precoiltNF −10 && ((qi1OF+qi2OF)/2.0)
    >3 "); // simulation for 3 sigma NR band cut with ptNF on x axis

TCut
    cQualityCuts=cLEChisq+cDeltaChisqGlitch+cDeltaChiSqLFN+cChargeChiSq+cSymmetricFV;
```

```
TCut upline("qi2OF< 0.88379*qi1OF + 1.89387");
TCut downline("qi2OF> 0.88379*qi1OF + 1.89387-2.75");
TCut symline=upline+downline;
TCut horizontal_line("qi2OF<1.89387");
TCut vertical_line("qi1OF<1.03231");
TCut flat_lines=horizontal_line+ vertical_line ;
TCut virajsymmetriccut=(symline)||(flat_lines);


cout<<"\nNumber of events: "<<chain2->GetEntries()<<endl;
cout<<"Name of Tree: "<<chain2->GetName()<<endl;

//chain2->Scan("SeriesNumber","SeriesNumber==11509070319", "colsize=15 precision=13");

double qi1OF_range_1=-2;
double qi1OF_range_2=10;
double qi2OF_range_1=-2;
double qi2OF_range_2=10;
double qo1OF_range_1=-2;
double qo1OF_range_2=5;
double qo2OF_range_1=-2;
double qo2OF_range_2=10;
double qsum1OF_range_1=-4;
double qsum1OF_range_2=15;
double qsum2OF_range_1=-5;
double qsum2OF_range_2=15;
double qi1OF_bins=100;
double qo1OF_bins=100;
double qi2OF_bins=100;
double qo2OF_bins=100;
double qsum1OF_bins=100;
double qsum2OF_bins=100;

gStyle->SetOptFit(1);

TH2D *h_charge = new TH2D("h_charge", Form("Z%d: Barium-%s, Inner Charge(S2) vs Inner
    Charge(S1)",zid, filename.c_str()), qi1OF_bins, qi1OF_range_1, qi1OF_range_2, qi2OF_bins,
    qi2OF_range_1, qi2OF_range_2);
h_charge->SetXTitle("Inner Charge (S1)");
h_charge->SetYTitle("Inner Charge (S2)");
h_charge->GetYaxis()->SetTitleOffset(1.5);
h_charge->GetYaxis()->SetLabelSize(0.03);
h_charge->GetYaxis()->SetTitleSize(0.03);
h_charge->SetMarkerColor(kRed);
h_charge->SetFillColor(kRed);
//h_charge->SetStats(kFALSE);

TCanvas *ch2d= new TCanvas("ch2d", "cTH2D inner charge(S2 vs S1)", 600, 600);
chain2->Draw("qi2OF:qi1OF>>h_charge",cBasicCuts);
//chain2->Draw("qo1OF:qi1OF>>h_charge");
```

```
//chain2->Draw("qo1OF:qi1OF>>h_charge",
    cNotEmpty+cRandom+cGoodFlash+cPstd+cBaseTemp+cVoltageBias, "same");
//ch2d->SaveAs(Form("%s/OuterCharge vs Innercharge_side1_%s_%s_%s.gif",
    opfileaddress.c_str(), fileaddress.c_str(),det.c_str(), weektitle.c_str()));

TF1 *gausfit=new TF1("gausfit","gaus", 3, 8);

TH1D *S1proj_onb, *S1proj_inevents;
double mu1,sigma1;

TCanvas* can1= new TCanvas("can1", "S2: Inner Charge Projection canvas", 600, 600);
S1proj_onb=h_charge->ProjectionY("S1proj_inevents", h_charge->GetXaxis()->FindBin(4),
    h_charge->GetXaxis()->FindBin(5));
cout<<" Side 1 charge X range " <<4<< " ----- "<<5<<endl;
//S1proj_onb->Rebin(10);
S1proj_onb->SetTitle(Form("Z%d: Barium-%s,%s, Side 2 Inner Charge Projection",zid,
    filename.c_str(), weektitle.c_str()));
S1proj_onb->SetYTitle("counts");
S1proj_onb->GetYaxis()->SetTitleOffset(1.5);
S1proj_onb->GetYaxis()->SetLabelSize(0.03);
S1proj_onb->GetYaxis()->SetTitleSize(0.03);
S1proj_onb->SetAxisRange(3, 8,"X");
//S1proj_onb->SetStats(kFALSE);
S1proj_onb->Draw();
//cout<<" Bin 0: "<<h_charge->GetXaxis()->FindBin(2)<<" Bin 1: "<<
    h_charge->GetXaxis()->FindBin(4)<<endl;
S1proj_onb->Fit("gausfit");
mu1=gausfit->GetParameter(1); sigma1=gausfit->GetParameter(2);
cout<<" x mean"<<4.5<<endl;
cout<<" mean + sigma= "<<gausfit->GetParameter(1)+gausfit->GetParameter(2)<<endl;
cout<<" mean + 2sigma=
    "<<gausfit->GetParameter(1)+2*gausfit->GetParameter(2)<<endl;


TH1D *S1proj_onb2, *S1proj_inevents2;
double mu2,sigma2;
TCanvas* can2= new TCanvas("can2", "S2: Inner Charge Projection canvas2", 600, 600);
S1proj_onb2=h_charge->ProjectionY("S1proj_inevents2", h_charge->GetXaxis()->FindBin(5),
    h_charge->GetXaxis()->FindBin(6));
cout<<" Side 1 charge X range " <<5<< " ----- "<<6<<endl;
//S1proj_onb->Rebin(10);
S1proj_onb2->SetTitle(Form("Z%d: Barium-%s,%s, Side 2 Inner Charge Projection",zid,
    filename.c_str(), weektitle.c_str()));
S1proj_onb2->SetYTitle("counts");
S1proj_onb2->GetYaxis()->SetTitleOffset(1.5);
S1proj_onb2->GetYaxis()->SetLabelSize(0.03);
S1proj_onb2->GetYaxis()->SetTitleSize(0.03);
S1proj_onb2->SetAxisRange(3, 8,"X");
//S1proj_onb->SetStats(kFALSE);
S1proj_onb2->Draw();
```

```
//cout<<" Bin 0: "<<h_charge->GetXaxis()->FindBin(2)<<" Bin 1: "<<
    h_charge->GetXaxis()->FindBin(4)<<endl;
S1proj_onb2->Fit("gausfit");
mu2=gausfit->GetParameter(1); sigma2=gausfit->GetParameter(2);

cout<<" x mean"<<5.5<<endl;
cout<<" mean + sigma= "<<gausfit->GetParameter(1)+gausfit->GetParameter(2)<<endl;
cout<<" mean + 2sigma=
    "<<gausfit->GetParameter(1)+2*gausfit->GetParameter(2)<<endl;



TCanvas* can_final_symcut= new TCanvas("can_final_symcut", "Inner Charge (S2 vs S1) with
    cut", 600, 600);
h_charge->Draw();
double slope1= (mu2+sigma2-mu1-sigma1)/1.0; cout<<"slope1 ="<<slope1<<endl;
double c1= mu2+sigma2 - slope1*5.5; cout<<"c1 ="<<c1<<endl;

TLine *line1 = new TLine(0,slope1*0+c1,7,slope1*7+c1);
line1->SetLineWidth(4);
line1->SetLineColor(3);
line1->Draw("same");
// 2.75 was -c to the 1st line  ... i.e bring the 1st line down parallelly

TLine *linedown = new TLine(-slope1/(c1-2.75),0,7,slope1*7+c1-2.75);
linedown->SetLineWidth(4);
linedown->SetLineColor(3);
linedown->Draw("same");

TLine *line_horizontal = new TLine(0,slope1*0+c1,-2,slope1*0+c1);
line_horizontal->SetLineWidth(4);
line_horizontal->SetLineColor(3);
line_horizontal->Draw("same");

cout<<"horizontal line = "<< c1<<endl;

TLine * line_vertical = new TLine(-slope1/(c1-2.75),0,-slope1/(c1-2.75),-2);
line_vertical->SetLineWidth(4);
line_vertical->SetLineColor(3);
line_vertical->Draw("same");

cout<<" vertical line = "<<-slope1/(c1-2.75)<<endl;

TLegend *legcut = new TLegend(0.5365772,0.7801394,0.6708054,0.8797909,NULL,"brNDC");
legcut->SetBorderSize(0);
legcut->SetTextSize(0.03);
legcut->SetLineColor(1);
legcut->SetLineStyle(1);
legcut->SetLineWidth(1);
legcut->SetFillColor(0);
```

```
legcut->SetFillStyle(0);
TLegendEntry *entryc=legcut->AddEntry("line1","Charge Symmetric cut ","lpf");
legcut->Draw("same");
entryc->SetTextFont(42);
entryc->SetMarkerColor(3);
entryc->SetMarkerStyle(1);
entryc->SetMarkerSize(3);
entryc->SetTextFont(42);
// TLine *line2 = new TLine(4.5,mu1-sigma1,5.5,mu2-sigma2);
//       line2->SetLineWidth(4);
//       line2->SetLineColor(5);
//       line2->Draw("same");

TCanvas *ch2d_check= new TCanvas("ch2d_check", "cTH2D inner charge(S2 vs S1) with
    Symmetric cut", 600, 600);
//chain2->Draw("qi2OF:qi1OF",cBasicCuts+virajsymmetriccut && "qi1OF>-2" &&
    "qi1OF<10" && "qi2OF>-2" && "qi2OF<10");
chain2->Draw("qi2OF:qi1OF",cBasicCuts && "qi1OF>-2" && "qi1OF<10" && "qi2OF>-2"
    && "qi2OF<10");
line1->Draw("same");
linedown->Draw("same");
 line_horizontal ->Draw("same");
 line_vertical ->Draw("same");
legcut->Draw("same");



}
```

8.4.7 ML Training ER and NR datasets.

```
/// \file
/// \ingroup tutorial_tmva
/// \notebook -nodraw
/// This macro provides examples for the training and testing of the
/// TMVA classifiers.
///
/// As input data is used a toy-MC sample consisting of four Gaussian-distributed
/// and linearly correlated input variables.
/// The methods to be used can be switched on and off by means of booleans, or
/// via the prompt command, for example:
///
///     root -l ./RRQtrain.C\(\"BDT\"\)
///     root -l ./RRQtrain.C\(\"BDTG\"\)
///    root
///
/// (note that the backslashes are mandatory)
/// If no method given, a default set of classifiers is used.
```

```
/// The output file "TMVA.root" can be analysed with the use of dedicated
/// macros (simply say: root -l <macro.C>), which can be conveniently
/// invoked through a GUI that will appear at the end of the RRQtrain of this macro.
/// Launch the GUI via the command:
///
///      root -l ./TMVAGui.C
///
/// You can also compile and RRQtrain the example with the following commands
///
///      make
///      ./RRQtrain <Methods>
///
/// where: '<Methods> = "method1 method2"' are the TMVA classifier names
/// example:
///
///      ./RRQtrain Fisher LikelihoodPCA BDT
///
/// If no method given, a default set is of classifiers is used
///
/// - Project   : TMVA - a ROOT-integrated toolkit for multivariate data analysis
/// - Package   : TMVA
/// - Root Macro: RRQtrain
///
/// \macro_output
/// \macro_code
/// \author Andreas Hoecker


#include <cstdlib>
#include <iostream>
#include <map>
#include <string>

#include "TChain.h"
#include "TFile.h"
#include "TTree.h"
#include "TString.h"
#include "TObjString.h"
#include "TSystem.h"
#include "TROOT.h"

#include "TMVA/Factory.h"
#include "TMVA/DataLoader.h"
#include "TMVA/Tools.h"
#include "TMVA/TMVAGui.h"

int RRQtrain( TString myMethodList = "" )
{
// The explicit loading of the shared libTMVA is done in TMVAlogon.C, defined in .rootrc
// if you use your private .rootrc, or run from a different directory, please copy the
// corresponding lines from .rootrc
```

```
// Methods to be processed can be given as an argument; use format:
//
//       mylinux~> root -l RRQtrain.C\(\"myMethod1,myMethod2,myMethod3\"\)

//---------------------------------------------------------------
// This loads the library
TMVA::Tools::Instance();

// Default MVA methods to be trained + tested
std::map<std::string,int> Use;

// Cut optimisation
Use["Cuts"]            = 1;
Use["CutsD"]           = 1;
Use["CutsPCA"]         = 0;
Use["CutsGA"]          = 0;
Use["CutsSA"]          = 0;
//
// 1-dimensional likelihood ("naive Bayes estimator")
Use["Likelihood"]      = 1;
Use["LikelihoodD"]     = 0; // the "D" extension indicates decorrelated input variables (see
    option strings)
Use["LikelihoodPCA"]   = 1; // the "PCA" extension indicates PCA-transformed input variables
    (see option strings)
Use["LikelihoodKDE"]   = 0;
Use["LikelihoodMIX"]   = 0;
//
// Mutidimensional likelihood and Nearest-Neighbour methods
Use["PDERS"]           = 1;
Use["PDERSD"]          = 0;
Use["PDERSPCA"]        = 0;
Use["PDEFoam"]         = 1;
Use["PDEFoamBoost"]    = 0; // uses generalised MVA method boosting
Use["KNN"]             = 1; // k-nearest neighbour method
//
// Linear Discriminant Analysis
Use["LD"]              = 1; // Linear Discriminant identical to Fisher
Use["Fisher"]          = 0;
Use["FisherG"]         = 0;
Use["BoostedFisher"]   = 0; // uses generalised MVA method boosting
Use["HMatrix"]         = 0;
//
// Function Discriminant analysis
Use["FDA_GA"]          = 1; // minimisation of user-defined function using Genetics Algorithm
Use["FDA_SA"]          = 0;
Use["FDA_MC"]          = 0;
Use["FDA_MT"]          = 0;
Use["FDA_GAMT"]        = 0;
Use["FDA_MCMT"]        = 0;
//
```

```
// Neural Networks (all are feed-forward Multilayer Perceptrons)
Use["MLP"]             = 0; // Recommended ANN
Use["MLPBFGS"]         = 0; // Recommended ANN with optional training method
Use["MLPBNN"]          = 1; // Recommended ANN with BFGS training method and bayesian
    regulator
Use["CFMlpANN"]        = 0; // Depreciated ANN from ALEPH
Use["TMlpANN"]         = 0; // ROOT's own ANN
Use["DNN_GPU"]         = 0; // CUDA-accelerated DNN training.
Use["DNN_CPU"]         = 0; // Multi-core accelerated DNN.
//
// Support Vector Machine
Use["SVM"]             = 1;
//
// Boosted Decision Trees
Use["BDT"]             = 1; // uses Adaptive Boost
Use["BDTG"]            = 0; // uses Gradient Boost
Use["BDTB"]            = 0; // uses Bagging
Use["BDTD"]            = 0; // decorrelation + Adaptive Boost
Use["BDTF"]            = 0; // allow usage of fisher discriminant for node splitting
//
// Friedman's RuleFit method, ie, an optimised series of cuts ("rules")
Use["RuleFit"]         = 1;
// ---------------------------------------------------------------

std::cout << std::endl;
std::cout << "==> Start RRQtrain" << std::endl;

// Select methods (don't look at this code - not of interest)
if (myMethodList != "") {
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) it->second
    = 0;

std::vector<TString> mlist = TMVA::gTools().SplitString( myMethodList, ',' );
for (UInt_t i=0; i<mlist.size(); i++) {
std::string regMethod(mlist[i]);

if (Use.find(regMethod) == Use.end()) {
std::cout << "Method \"" << regMethod << "\" not known in TMVA under this name.
    Choose among the following:" << std::endl;
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) std::cout
    << it->first << " ";
std::cout << std::endl;
return 1;
}
Use[regMethod] = 1;
}
}

string featurename1,featurename2,featurename3,featurename4,
featurename5,featurename6,featurename7;
```

```
featurename1="precoiltNF";
featurename2="ytNF";
featurename3="qzpartOF";
featurename4="qrpart1OF";
featurename5="pzpartOF";
featurename6="prpartOF";
featurename7="qsummaxOF";
// ------------------------------------------------------------

// Here the preparation phase begins

// Read training and test data
// (it is also possible to use ASCII format as input -> see TMVA Users Guide)


TCut upline("qi2OF< 0.88379*qi1OF + 1.89387");
TCut downline("qi2OF> 0.88379*qi1OF + 1.89387-2.75");
TCut symline=upline+downline;
TCut horizontal_line("qi2OF<1.89387");
TCut vertical_line("qi1OF<1.03231");
TCut flat_lines=horizontal_line+ vertical_line ;
TCut virajsymmetriccut=(symline)||(flat_lines);



TCut virajPhononRadial("prpartOF<(TMath::Exp(-0.424675-3.53768*ptNF)+0.295494)"); //
    2 sigma
TCut virajchargeradialcut("qo1OF<1.05087 && qo2OF<1.23711");

TCut threesigmacut("( (qi1OF+qi2OF)/2.0 ) > 0.95*precoiltNF -10 && ((qi1OF+qi2OF)/2.0)
    >3 ");

TCut newERNRcut("( (qi1OF+qi2OF)/2.0 ) > 0.55*precoiltNF +1.5 ");

TCut qimeancut("0.5*(qi1OF+qi2OF)>0");
TCut ytNFNRcut("ytNF<0.6");

TCut mycuts = qimeancut+!newERNRcut; // for example: TCut mycuts = "abs(var1)<0.5 &&
    abs(var2-0.5)<1";
TCut mycutb = qimeancut+newERNRcut; // for example: TCut mycutb = "abs(var1)<0.5";

//TFile *sigSrc = new TFile("WIMPmodel_RRQ_cf.root","READ");
TFile *sigSrc = new TFile("RRQ_NR_cf.root","READ");
TTree* signalTree = (TTree*)sigSrc->Get( "tree_bkg" );

signalTree->Scan();
TCanvas* can1= new TCanvas("can1", "qimeans vs precoiltNF", 600, 600);
can1->cd();
signalTree->Draw("qimean:precoiltNF",qimeancut && !newERNRcut);

/* TCanvas* can2= new TCanvas("can2", "qimeans vs precoiltNF with FV cut", 600, 600);
```

```
can2->cd();
signalTree->Draw("qimean:precoiltNF","qimean>0" && virajchargeradialcut);
*/
//TFile *bkgSrc = new TFile("gammamodel_RRQ_ba_no3sigma_w.root","READ");
TCanvas* canbkg= new TCanvas("canbkg", "qimeans vs precoiltNF Ba events", 600, 600);
TFile *bkgSrc = new TFile("RRQ_ER.root","READ");
TTree* backgroundtree = (TTree*)bkgSrc->Get( "tree_bkg" );
// TTree* gamma_tree_weight_precoiltNF=(TTree*)bkgSrc->Get( "tree_weight_precoiltNF" );
//backgroundtree->Scan();
backgroundtree->Draw("qimean:precoiltNF",qimeancut && newERNRcut);

TString outfileName( "TMVA.root" );
TFile* outputFile = TFile::Open( outfileName, "RECREATE" );

// Create the factory object. Later you can choose the methods
// whose performance you'd like to investigate. The factory is
// the only TMVA object you have to interact with
//
// The first argument is the base of the name of all the
// weightfiles in the directory weight/
//
// The second argument is the output file for the training results
// All TMVA output can be suppressed by removing the "!" (not) in
// front of the "Silent" argument in the option string
// TMVA::Factory *factory = new TMVA::Factory( "RRQtrain", outputFile,
//
    "!V:!Silent:Color:DrawProgressBar:Transformations=I;D;P;G,D:AnalysisType=Classification"
    );
TMVA::Factory *factory = new TMVA::Factory( "RRQtrain", outputFile,
"!V:!Silent:Color:DrawProgressBar:Transformations=I;D;P;G,D:AnalysisType=Classification" );

TMVA::DataLoader *dataloader=new TMVA::DataLoader("dataset");
// If you wish to modify default settings
// (please check "src/Config.h" to see all available global options)
//
//    (TMVA::gConfig().GetVariablePlotting()).fTimesRMS = 8.0;
//   (TMVA::gConfig().GetIONames()).fWeightFileDir = "myWeightDirectory";

// Define the input variables that shall be used for the MVA training
// note that you may also use variable expressions, such as: "3*var1/var2*abs(var3)"
// [all types of expressions that can also be parsed by TTree::Draw( "expression" )]


//double precoiltNF,ytNF, qzpartOF,qrpart1OF,pzpartOF,prpartOF,qsummaxOF;

dataloader->AddVariable( "precoiltNF","precoiltNF", " keV",'F' );
dataloader->AddVariable( "ytNF","ytNF", " ",'F' );
dataloader->AddVariable( "qzpartOF", "qzpartOF","", 'F' );
dataloader->AddVariable( "qrpart1OF", "qrpart1OF","", 'F' );
dataloader->AddVariable( "pzpartOF", "pzpartOF","", 'F' );
dataloader->AddVariable( "prpartOF", "prpartOF","", 'F' );
```

```
dataloader->AddVariable( "qsummaxOF", "qsummaxOF","", 'F' );



// You can add so-called "Spectator variables", which are not used in the MVA training,
// but will appear in the final "TestTree" produced by TMVA. This TestTree will contain the
// input variables, the response values of all trained MVAs, and the spectator variables


// global event weights per tree (see below for setting event-wise weights)
Double_t signalWeight     = 1;  //
Double_t backgroundWeight =1; //

// You can add an arbitrary number of signal or background trees
dataloader->AddSignalTree ( signalTree,     signalWeight );
dataloader->AddBackgroundTree( backgroundtree, backgroundWeight );
// To give different trees for training and testing, do as follows:
//
//     dataloader->AddSignalTree( signalTrainingTree, signalTrainWeight, "Training" );
//     dataloader->AddSignalTree( signalTestTree,  signalTestWeight,  "Test" );

// Use the following code instead of the above two or four lines to add signal and background
// training and test events "by hand"
// NOTE that in this case one should not give expressions (such as "var1+var2") in the input
//      variable definition, but simply compute the expression before adding the event
// ``` cpp
// // --- begin ----------------------------------------------------
// std::vector<Double_t> vars( 4 ); // vector has size of number of input variables
// Float_t  treevars[4], weight;
//
// // Signal
// for (UInt_t ivar=0; ivar<4; ivar++) signalTree->SetBranchAddress( Form( "var%i", ivar+1
    ), &(treevars[ivar]) );
// for (UInt_t i=0; i<signalTree->GetEntries(); i++) {
//    signalTree->GetEntry(i);
//    for (UInt_t ivar=0; ivar<4; ivar++) vars[ivar] = treevars[ivar];
//    // add training and test events; here: first half is training, second is testing
//    // note that the weight can also be event-wise
//    if (i < signalTree->GetEntries()/2.0) dataloader->AddSignalTrainingEvent( vars,
    signalWeight );
//    else                                  dataloader->AddSignalTestEvent ( vars, signalWeight
    );
// }
//
// // Background (has event weights)
// background->SetBranchAddress( "weight", &weight );
// for (UInt_t ivar=0; ivar<4; ivar++) background->SetBranchAddress( Form( "var%i",
    ivar+1 ), &(treevars[ivar]) );
// for (UInt_t i=0; i<background->GetEntries(); i++) {
```

```
//     background->GetEntry(i);
//     for (UInt_t ivar=0; ivar<4; ivar++) vars[ivar] = treevars[ivar];
//     // add training and test events; here: first half is training, second is testing
//     // note that the weight can also be event-wise
//     if (i < background->GetEntries()/2) dataloader->AddBackgroundTrainingEvent( vars,
    backgroundWeight*weight );
//     else                                dataloader->AddBackgroundTestEvent ( vars,
    backgroundWeight*weight );
// }
// // --- end ----------------------------------------------------------
// ```
// End of tree registration

/*
double precoiltNFrange_1=0;
double precoiltNFrange_2=170;
int precoiltNFbinsize=450;
int qbinsize=500; // Prev: 500, 160
double q_y_range_1=0; //Prev: -10, -2
double q_y_range_2=160; //Prev: 80, 14

TH2D *hERNR_bkg = new TH2D("hERNR_bkg","ER NR", precoiltNFbinsize,
    precoiltNFrange_1, precoiltNFrange_2, qbinsize, q_y_range_1, q_y_range_2);
hERNR_bkg->SetXTitle("precoiltNF [keV]");
hERNR_bkg->SetYTitle("qimean");
hERNR_bkg->GetYaxis()->SetTitleOffset(1.5);
hERNR_bkg->GetYaxis()->SetLabelSize(0.03);
hERNR_bkg->GetYaxis()->SetTitleSize(0.03);


float_t precoiltNF_gamma,ytNF_gamma,
    qzpartOF_gamma,qrpart1OF_gamma,pzpartOF_gamma,prpartOF_gamma,qsummaxOF_gamma;

float_t PTNFchisq_gamma,
ptNF_gamma,
PTOFchisq_gamma,
PTglitch1OFchisq_gamma,
PTlfnoise1OFchisq_gamma,
QS1OFchisq_gamma,
qsum1OF_gamma,
QS2OFchisq_gamma,
qsum2OF_gamma,
qo1OF_gamma,
qo2OF_gamma,
qi1OF_gamma,
qi2OF_gamma,
qimean_gamma;

std::vector<Double_t> gammavars( 7 ); // vector has size of number of input variables
Float_t gammatreevars[7];
```

```
backgroundtree->SetBranchAddress("PTNFchisq",&PTNFchisq_gamma);
backgroundtree->SetBranchAddress("ptNF",&ptNF_gamma);
backgroundtree->SetBranchAddress("PTOFchisq",&PTOFchisq_gamma);
backgroundtree->SetBranchAddress("PTglitch1OFchisq",&PTglitch1OFchisq_gamma);
backgroundtree->SetBranchAddress("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_gamma);
backgroundtree->SetBranchAddress("QS1OFchisq",&QS1OFchisq_gamma);
backgroundtree->SetBranchAddress("qsum1OF",&qsum1OF_gamma);
backgroundtree->SetBranchAddress("QS2OFchisq",&QS2OFchisq_gamma);
backgroundtree->SetBranchAddress("qsum2OF",&qsum2OF_gamma);
backgroundtree->SetBranchAddress("qo1OF",&qo1OF_gamma);
backgroundtree->SetBranchAddress("qo2OF",&qo2OF_gamma);
backgroundtree->SetBranchAddress("qi1OF",&qi1OF_gamma);


backgroundtree->SetBranchAddress("qi2OF",&qi2OF_gamma);


backgroundtree->SetBranchAddress("qimean",&qimean_gamma);


backgroundtree->SetBranchAddress("precoiltNF",&precoiltNF_gamma);
backgroundtree->SetBranchAddress("ytNF", &ytNF_gamma);
backgroundtree->SetBranchAddress("qzpartOF", &qzpartOF_gamma);
backgroundtree->SetBranchAddress("qrpart1OF", &qrpart1OF_gamma);
backgroundtree->SetBranchAddress("pzpartOF", &pzpartOF_gamma);
backgroundtree->SetBranchAddress("prpartOF", &prpartOF_gamma);
backgroundtree->SetBranchAddress("qsummaxOF", &qsummaxOF_gamma);

//backgroundtree->SetBranchAddress("precoiltNF",&(gammatreevars[0]));
//backgroundtree->SetBranchAddress("ytNF", &(gammatreevars[1]));
//backgroundtree->SetBranchAddress("qzpartOF", &(gammatreevars[2]));
//backgroundtree->SetBranchAddress("qrpart1OF", &(gammatreevars[3]));
//backgroundtree->SetBranchAddress("pzpartOF", &(gammatreevars[4]));
//backgroundtree->SetBranchAddress("prpartOF", &(gammatreevars[5]));
//backgroundtree->SetBranchAddress("qsummaxOF", &(gammatreevars[6]));



int  nevents=backgroundtree->GetEntries();
cout<<"nevents : "<<nevents<<endl;
for  (Long64_t ievt=0; ievt<nevents;ievt++){
backgroundtree->GetEntry(ievt);
hERNR_bkg->Fill(precoiltNF_gamma,qimean_gamma);
}
TCanvas *can;
can = new TCanvas("can","",10,10,600,600);
can->SetLeftMargin(0.2);
can->SetRightMargin(0.05);
can->SetTopMargin(0.05);
can->SetBottomMargin(0.13);
can->cd();
```

```
hERNR_bkg->GetXaxis()->SetTitle("precoiltNF (keV)");
hERNR_bkg->GetXaxis()->SetTitleOffset(1.2);
hERNR_bkg->GetYaxis()->SetTitleOffset(1.7);
hERNR_bkg->GetYaxis()->SetTitle("qimean");
hERNR_bkg->SetMarkerColor(2); //2=Red
hERNR_bkg->SetMarkerStyle(15);
hERNR_bkg->SetMarkerSize(1.0);
hERNR_bkg->Draw();

TH2D *hERNR_sig = new TH2D("hERNR_sig","ER NR", precoiltNFbinsize, precoiltNFrange_1,
    precoiltNFrange_2, qbinsize, q_y_range_1, q_y_range_2);
hERNR_sig->SetXTitle("precoiltNF [keV]");
hERNR_sig->SetYTitle("qimean");
hERNR_sig->GetYaxis()->SetTitleOffset(1.5);
hERNR_sig->GetYaxis()->SetLabelSize(0.03);
hERNR_sig->GetYaxis()->SetTitleSize(0.03);


 float_t  precoiltNF_signal,ytNF_signal,
    qzpartOF_signal,qrpart1OF_signal,pzpartOF_signal,prpartOF_signal,qsummaxOF_signal;
 float_t  PTNFchisq_signal,
ptNF_signal,
PTOFchisq_signal,
PTglitch1OFchisq_signal,
PTlfnoise1OFchisq_signal,
QS1OFchisq_signal,
qsum1OF_signal,
QS2OFchisq_signal,
qsum2OF_signal,
qo1OF_signal,
qo2OF_signal,
qi1OF_signal,
qi2OF_signal,
qimean_signal;


signalTree->SetBranchAddress("PTNFchisq",&PTNFchisq_signal);
signalTree->SetBranchAddress("ptNF",&ptNF_signal);
signalTree->SetBranchAddress("PTOFchisq",&PTOFchisq_signal);
signalTree->SetBranchAddress("PTglitch1OFchisq",&PTglitch1OFchisq_signal);
signalTree->SetBranchAddress("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_signal);
signalTree->SetBranchAddress("QS1OFchisq",&QS1OFchisq_signal);
signalTree->SetBranchAddress("qsum1OF",&qsum1OF_signal);
signalTree->SetBranchAddress("QS2OFchisq",&QS2OFchisq_signal);
signalTree->SetBranchAddress("qsum2OF",&qsum2OF_signal);
signalTree->SetBranchAddress("qo1OF",&qo1OF_signal);
signalTree->SetBranchAddress("qo2OF",&qo2OF_signal);
signalTree->SetBranchAddress("qi1OF",&qi1OF_signal);
```

```
signalTree->SetBranchAddress("qi2OF",&qi2OF_signal);


signalTree->SetBranchAddress("qimean",&qimean_signal);


signalTree->SetBranchAddress("precoiltNF",&precoiltNF_signal);
signalTree->SetBranchAddress("ytNF", &ytNF_signal);
signalTree->SetBranchAddress("qzpartOF", &qzpartOF_signal);
signalTree->SetBranchAddress("qrpart1OF", &qrpart1OF_signal);
signalTree->SetBranchAddress("pzpartOF", &pzpartOF_signal);
signalTree->SetBranchAddress("prpartOF", &prpartOF_signal);
signalTree->SetBranchAddress("qsummaxOF", &qsummaxOF_signal);


int  nevents_sig=signalTree->GetEntries();
cout<<"nevents_sig : "<<nevents_sig<<endl;
for  (Long64_t ievt=0; ievt<nevents_sig;ievt++){
signalTree->GetEntry(ievt);
hERNR_sig->Fill(precoiltNF_signal,qimean_signal);
}
TCanvas *can2;
can2 = new TCanvas("can2","",10,10,600,600);
can2->SetLeftMargin(0.2);
can2->SetRightMargin(0.05);
can2->SetTopMargin(0.05);
can2->SetBottomMargin(0.13);
can2->cd();



hERNR_sig->GetXaxis()->SetTitle("precoiltNF (keV)");
hERNR_sig->GetXaxis()->SetTitleOffset(1.2);
hERNR_sig->GetYaxis()->SetTitleOffset(1.7);
hERNR_sig->GetYaxis()->SetTitle("qimean");
hERNR_sig->SetMarkerColor(2); //2=Red
hERNR_sig->SetMarkerStyle(15);
hERNR_sig->SetMarkerSize(1.0);
hERNR_sig->Draw();


*/



/* for  (Long64_t ievt=0; ievt<nevents;ievt++)
{
// cout<<ievt<<endl;
if  (ievt%10000 == 0)
```

```
std::cout << "---      .....      Processing event:   " << ievt << std::endl;
backgroundtree->GetEntry(ievt);
for (UInt_t ivar=0; ivar<7; ivar++) gammavars[ivar] = gammatreevars[ivar];
//gammavars[0]=precoiltNF_gamma;
//gammavars[1]=ytNF_gamma;
//gammavars[2]=qzpartOF_gamma;
//gammavars[3]=qrpart1OF_gamma;
//gammavars[4]=pzpartOF_gamma;
//gammavars[5]=prpartOF_gamma;
//gammavars[6]=qsummaxOF_gamma;


if (ievt%2==0) dataloader->AddBackgroundTrainingEvent( gammavars, 1 );
else           dataloader->AddBackgroundTestEvent( gammavars, 1 );



}
*/


// Set individual event weights (the variables must exist in the original TTree)
// -   for signal     : `dataloader->SetSignalWeightExpression ("weight1*weight2");`
// -   for background: `dataloader->SetBackgroundWeightExpression("weight1*weight2");`
//dataloader->SetBackgroundWeightExpression( "weight" );
//dataloader->SetSignalWeightExpression("8900581/138287.0");
//dataloader->SetBackgroundWeightExpression("8900581/5875287.0");
// Apply additional cuts on the signal and background samples (can be different)
//TCut mycuts = ""; // for example: TCut mycuts = "abs(var1)<0.5 && abs(var2-0.5)<1";
// TCut mycutb = "0.7<invmass<1.1"; // for example: TCut mycutb = "abs(var1)<0.5";



// Tell the dataloader how to use the training and testing events
//
// If no numbers of events are given, half of the events in the tree are used
// for training, and the other half for testing:
//
//    dataloader->PrepareTrainingAndTestTree( mycut, "SplitMode=random:!V" );
//
// To also specify the number of testing events, use:
//
//    dataloader->PrepareTrainingAndTestTree( mycut,
//
    "NSigTrain=3000:NBkgTrain=3000:NSigTest=3000:NBkgTest=3000:SplitMode=Random:!V"
    );
//dataloader->PrepareTrainingAndTestTree( mycuts, mycutb,
//
    "nTrain_Signal=0:nTrain_Background=100000:nTest_Background=100000:
//SplitMode=Random:NormMode=NumEvents:!V" );
//dataloader->PrepareTrainingAndTestTree( mycuts, mycutb,
```

```
//     "nTrain_Signal=50000:nTest_Signal=50000:nTrain_Background=100000:
//nTest_Background=100000:SplitMode=Random:NormMode=NumEvents:!V" );
dataloader->PrepareTrainingAndTestTree( mycuts, mycutb,
"nTrain_Signal=0 :SplitMode=Random:NormMode=NumEvents:!V" );
// ### Book MVA methods
//
// Please lookup the various method configuration options in the corresponding cxx files , eg:
// src/MethoCuts.cxx, etc, or here: http://tmva.sourceforge.net/optionRef.html
// it is possible to preset ranges in the option string in which the cut optimisation should be
     done:
// "...: CutRangeMin[2]=-1:CutRangeMax[2]=1"...", where [2] is the third input variable

// Cut optimisation
if (Use["Cuts"])
factory->BookMethod( dataloader, TMVA::Types::kCuts, "Cuts",
"!H:!V:FitMethod=MC:EffSel:SampleSize=200000:VarProp=FSmart" );

if (Use["CutsD"])
factory->BookMethod( dataloader, TMVA::Types::kCuts, "CutsD",
"!H:!V:FitMethod=MC:EffSel:SampleSize=200000:VarProp=FSmart:VarTransform=Decorrelate"
     );

if (Use["CutsPCA"])
factory->BookMethod( dataloader, TMVA::Types::kCuts, "CutsPCA",
"!H:!V:FitMethod=MC:EffSel:SampleSize=200000:VarProp=FSmart:VarTransform=PCA" );

if (Use["CutsGA"])
factory->BookMethod( dataloader, TMVA::Types::kCuts, "CutsGA",
"H:!V:FitMethod=GA:CutRangeMin[0]=-10:CutRangeMax[0]=10:VarProp[1]=
FMax:EffSel:Steps=30:Cycles=3:PopSize=400:SC_steps=10:SC_rate=5:SC_factor=0.95" );

if (Use["CutsSA"])
factory->BookMethod( dataloader, TMVA::Types::kCuts, "CutsSA",
"!H:!V:FitMethod=SA:EffSel:MaxCalls=150000:KernelTemp=IncAdaptive:InitialTemp=
1e+6:MinTemp=1e-6:Eps=1e-10:UseDefaultScale" );

// Likelihood ("naive Bayes estimator")
if (Use["Likelihood"])
factory->BookMethod( dataloader, TMVA::Types::kLikelihood, "Likelihood",
"H:!V:TransformOutput:PDFInterpol=Spline2:NSmoothSig[0]=20:NSmoothBkg[0]=
20:NSmoothBkg[1]=10:NSmooth=1:NAvEvtPerBin=50" );

// Decorrelated likelihood
if (Use["LikelihoodD"])
factory->BookMethod( dataloader, TMVA::Types::kLikelihood, "LikelihoodD",
"!H:!V:TransformOutput:PDFInterpol=Spline2:NSmoothSig[0]=20:NSmoothBkg[0]
=20:NSmooth=5:NAvEvtPerBin=50:VarTransform=Decorrelate" );

// PCA-transformed likelihood
if (Use["LikelihoodPCA"])
factory->BookMethod( dataloader, TMVA::Types::kLikelihood, "LikelihoodPCA",
```

```
"!H:!V:!TransformOutput:PDFInterpol=Spline2:NSmoothSig[0]=20:NSmoothBkg[0]
=20:NSmooth=5:NAvEvtPerBin=50:VarTransform=PCA" );

// Use a kernel density estimator to approximate the PDFs
if (Use["LikelihoodKDE"])
factory->BookMethod( dataloader, TMVA::Types::kLikelihood, "LikelihoodKDE",
"!H:!V:!TransformOutput:PDFInterpol=KDE:KDEtype=Gauss:KDEiter=
Adaptive:KDEFineFactor=0.3:KDEborder=None:NAvEvtPerBin=50" );

// Use a variable-dependent mix of splines and kernel density estimator
if (Use["LikelihoodMIX"])
factory->BookMethod( dataloader, TMVA::Types::kLikelihood, "LikelihoodMIX",
"!H:!V:!TransformOutput:PDFInterpolSig[0]=KDE:PDFInterpolBkg[0]=
KDE:PDFInterpolSig[1]=KDE:PDFInterpolBkg[1]=KDE:PDFInterpolSig[2]=
Spline2:PDFInterpolBkg[2]=Spline2:PDFInterpolSig[3]=Spline2:PDFInterpolBkg[3]=
Spline2:KDEtype=Gauss:KDEiter=Nonadaptive:KDEborder=None:NAvEvtPerBin=50" );

// Test the multi-dimensional probability density estimator
// here are the options strings for the MinMax and RMS methods, respectively:
//
//
    "!H:!V:VolumeRangeMode=MinMax:DeltaFrac=0.2:KernelEstimator=Gauss:GaussSigma=0.3"
    );
//
    "!H:!V:VolumeRangeMode=RMS:DeltaFrac=3:KernelEstimator=Gauss:GaussSigma=0.3" );
if (Use["PDERS"])
factory->BookMethod( dataloader, TMVA::Types::kPDERS, "PDERS",
"!H:!V:NormTree=T:VolumeRangeMode=Adaptive:KernelEstimator=Gauss:GaussSigma=0.3:
NEventsMin=400:NEventsMax=600" );

if (Use["PDERSD"])
factory->BookMethod( dataloader, TMVA::Types::kPDERS, "PDERSD",
"!H:!V:VolumeRangeMode=Adaptive:KernelEstimator=Gauss:GaussSigma=0.3:
NEventsMin=400:NEventsMax=600:VarTransform=Decorrelate" );

if (Use["PDERSPCA"])
factory->BookMethod( dataloader, TMVA::Types::kPDERS, "PDERSPCA",
"!H:!V:VolumeRangeMode=Adaptive:KernelEstimator=Gauss:GaussSigma=0.3:
NEventsMin=400:NEventsMax=600:VarTransform=PCA" );

// Multi-dimensional likelihood estimator using self-adapting phase-space binning
if (Use["PDEFoam"])
factory->BookMethod( dataloader, TMVA::Types::kPDEFoam, "PDEFoam",
"!H:!V:SigBgSeparate=F:TailCut=0.001:VolFrac=0.0666:nActiveCells=500:
nSampl=2000:nBin=5:Nmin=100:Kernel=None:Compress=T" );

if (Use["PDEFoamBoost"])
factory->BookMethod( dataloader, TMVA::Types::kPDEFoam, "PDEFoamBoost",
"!H:!V:Boost_Num=30:Boost_Transform=linear:SigBgSeparate=F:MaxDepth=4:
UseYesNoCell=T:DTLogic=MisClassificationError:FillFoamWithOrigWeights=F:
TailCut=0:nActiveCells=500:nBin=20:Nmin=400:Kernel=None:Compress=T" );
```

```
// K−Nearest Neighbour classifier (KNN)
if (Use["KNN"])
factory−>BookMethod( dataloader, TMVA::Types::kKNN, "KNN",
"H:nkNN=20:ScaleFrac=0.8:SigmaFact=1.0:Kernel=Gaus:UseKernel=F:UseWeight=T:!Trim" );

// H−Matrix (chi2−squared) method
if (Use["HMatrix"])
factory−>BookMethod( dataloader, TMVA::Types::kHMatrix, "HMatrix",
    "!H:!V:VarTransform=None" );

// Linear discriminant (same as Fisher discriminant)
if (Use["LD"])
factory−>BookMethod( dataloader, TMVA::Types::kLD, "LD",
    "H:!V:VarTransform=None:CreateMVAPdfs:PDFInterpolMVAPdf=Spline2:
NbinsMVAPdf=50:NsmoothMVAPdf=10" );

// Fisher discriminant (same as LD)
if (Use["Fisher"])
factory−>BookMethod( dataloader, TMVA::Types::kFisher, "Fisher",
    "H:!V:Fisher:VarTransform=None:CreateMVAPdfs:PDFInterpolMVAPdf=Spline2:
NbinsMVAPdf=50:NsmoothMVAPdf=10" );

// Fisher with Gauss−transformed input variables
if (Use["FisherG"])
factory−>BookMethod( dataloader, TMVA::Types::kFisher, "FisherG",
    "H:!V:VarTransform=Gauss" );

// Composite classifier: ensemble (tree) of boosted Fisher classifiers
if (Use["BoostedFisher"])
factory−>BookMethod( dataloader, TMVA::Types::kFisher, "BoostedFisher",
"H:!V:Boost_Num=20:Boost_Transform=log:Boost_Type=AdaBoost:Boost_AdaBoostBeta=0.2:
!Boost_DetailedMonitoring" );

// Function discrimination analysis (FDA) −− test of various fitters − the recommended one is
    Minuit (or GA or SA)
if (Use["FDA_MC"])
factory−>BookMethod( dataloader, TMVA::Types::kFDA, "FDA_MC",
"H:!V:Formula=(0)+(1)*x0+(2)*x1+(3)*x2+(4)*x3:ParRanges=(−1,1);(−10,10);(−10,10);
(−10,10);(−10,10):FitMethod=MC:SampleSize=100000:Sigma=0.1" );

if (Use["FDA_GA"]) // can also use Simulated Annealing (SA) algorithm (see Cuts_SA options])
factory−>BookMethod( dataloader, TMVA::Types::kFDA, "FDA_GA",
"H:!V:Formula=(0)+(1)*x0+(2)*x1+(3)*x2+(4)*x3:ParRanges=(−1,1);(−10,10);(−10,10);
(−10,10);(−10,10):FitMethod=GA:PopSize=100:Cycles=2:Steps=5:Trim=True:SaveBestGen=1"
    );

if (Use["FDA_SA"]) // can also use Simulated Annealing (SA) algorithm (see Cuts_SA options])
factory−>BookMethod( dataloader, TMVA::Types::kFDA, "FDA_SA",
"H:!V:Formula=(0)+(1)*x0+(2)*x1+(3)*x2+(4)*x3:ParRanges=(−1,1);(−10,10);(−10,10);
(−10,10);(−10,10):FitMethod=SA:MaxCalls=15000:KernelTemp=IncAdaptive:
```

```
InitialTemp=1e+6:MinTemp=1e-6:Eps=1e-10:UseDefaultScale" );

if (Use["FDA_MT"])
factory->BookMethod( dataloader, TMVA::Types::kFDA, "FDA_MT",
"H:!V:Formula=(0)+(1)*x0+(2)*x1+(3)*x2+(4)*x3:ParRanges=(-1,1);(-10,10);(-10,10);
(-10,10);(-10,10):FitMethod=MINUIT:ErrorLevel=1:PrintLevel=-1:FitStrategy=2:
UseImprove:UseMinos:SetBatch" );

if (Use["FDA_GAMT"])
factory->BookMethod( dataloader, TMVA::Types::kFDA, "FDA_GAMT",
"H:!V:Formula=(0)+(1)*x0+(2)*x1+(3)*x2+(4)*x3:ParRanges=(-1,1);(-10,10);(-10,10);
(-10,10);(-10,10):FitMethod=GA:Converger=MINUIT:ErrorLevel=1:
PrintLevel=-1:FitStrategy=0:!UseImprove:!UseMinos:SetBatch:Cycles=1:
PopSize=5:Steps=5:Trim" );

if (Use["FDA_MCMT"])
factory->BookMethod( dataloader, TMVA::Types::kFDA, "FDA_MCMT",
"H:!V:Formula=(0)+(1)*x0+(2)*x1+(3)*x2+(4)*x3:ParRanges=(-1,1);(-10,10);(-10,10);
(-10,10);(-10,10):FitMethod=MC:Converger=MINUIT:ErrorLevel=1:PrintLevel=-1:
FitStrategy=0:!UseImprove:!UseMinos:SetBatch:SampleSize=20" );

// TMVA ANN: MLP (recommended ANN) -- all ANNs in TMVA are Multilayer Perceptrons
if (Use["MLP"])
factory->BookMethod( dataloader, TMVA::Types::kMLP, "MLP",
    "H:!V:NeuronType=tanh:VarTransform=N:NCycles=600:HiddenLayers=N+5:TestRate=5:
!UseRegulator" );

if (Use["MLPBFGS"])
factory->BookMethod( dataloader, TMVA::Types::kMLP, "MLPBFGS",
    "H:!V:NeuronType=tanh:VarTransform=N:NCycles=600:HiddenLayers=N+5:TestRate=5:
TrainingMethod=BFGS:!UseRegulator" );

if (Use["MLPBNN"])
factory->BookMethod( dataloader, TMVA::Types::kMLP, "MLPBNN",
    "H:!V:NeuronType=tanh:VarTransform=N:NCycles=60:HiddenLayers=N+5:TestRate=5:
TrainingMethod=BFGS:UseRegulator" ); // BFGS training with bayesian regulators


// Multi-architecture DNN implementation.
if (Use["DNN_CPU"] or Use["DNN_GPU"]) {
// General layout.
TString layoutString ("Layout=TANH|128,TANH|128,TANH|128,LINEAR");

// Training strategies.
TString training0("LearningRate=1e-1,Momentum=0.9,Repetitions=1,"
"ConvergenceSteps=20,BatchSize=256,TestRepetitions=10,"
"WeightDecay=1e-4,Regularization=L2,"
"DropConfig=0.0+0.5+0.5+0.5, Multithreading=True");
TString training1("LearningRate=1e-2,Momentum=0.9,Repetitions=1,"
"ConvergenceSteps=20,BatchSize=256,TestRepetitions=10,"
"WeightDecay=1e-4,Regularization=L2,"
```

```
"DropConfig=0.0+0.0+0.0+0.0, Multithreading=True");
TString training2("LearningRate=1e-3,Momentum=0.0,Repetitions=1,"
"ConvergenceSteps=20,BatchSize=256,TestRepetitions=10,"
"WeightDecay=1e-4,Regularization=L2,"
"DropConfig=0.0+0.0+0.0+0.0, Multithreading=True");
TString trainingStrategyString ("TrainingStrategy=");
trainingStrategyString += training0 + "|" + training1 + "|" + training2;

// General Options.
TString dnnOptions ("!H:V:ErrorStrategy=CROSSENTROPY:VarTransform=N:"
" WeightInitialization=XAVIERUNIFORM");
dnnOptions.Append (":"); dnnOptions.Append (layoutString);
dnnOptions.Append (":"); dnnOptions.Append (trainingStrategyString);

// Cuda implementation.
if (Use["DNN_GPU"]) {
TString gpuOptions = dnnOptions + ":Architecture=GPU";
factory->BookMethod(dataloader, TMVA::Types::kDNN, "DNN_GPU", gpuOptions);
}
// Multi-core CPU implementation.
if (Use["DNN_CPU"]) {
TString cpuOptions = dnnOptions + ":Architecture=CPU";
factory->BookMethod(dataloader, TMVA::Types::kDNN, "DNN_CPU", cpuOptions);
}
}

// CF(Clermont-Ferrand)ANN
if (Use["CFMlpANN"])
factory->BookMethod( dataloader, TMVA::Types::kCFMlpANN, "CFMlpANN",
    "!H:!V:NCycles=200:HiddenLayers=N+1,N" ); // n_cycles:#nodes:#nodes:...

// Tmlp(Root)ANN
if (Use["TMlpANN"])
factory->BookMethod( dataloader, TMVA::Types::kTMlpANN, "TMlpANN",
    "!H:!V:NCycles=200:HiddenLayers=N+1,N:LearningMethod=BFGS:ValidationFraction=0.3"
    ); // n_cycles:#nodes:#nodes:...

// Support Vector Machine
if (Use["SVM"])
factory->BookMethod( dataloader, TMVA::Types::kSVM, "SVM",
    "Gamma=0.25:Tol=0.001:VarTransform=Norm" );

// Boosted Decision Trees************************************
if (Use["BDTG"]) // Gradient Boost
factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDTG",
"!H:!V:NTrees=1000:MinNodeSize=2.5%:BoostType=Grad:Shrinkage=0.10:
UseBaggedBoost:BaggedSampleFraction=0.5:nCuts=20:MaxDepth=2" );
//************************************************************
/* if (Use["BDT"]) // Adaptive Boost
factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDT",
"!H:!V:NTrees=850:MinNodeSize=2.5%:MaxDepth=3:BoostType=AdaBoost:
```

```
AdaBoostBeta=0.5:UseBaggedBoost:BaggedSampleFraction=0.5:
SeparationType=GiniIndex:nCuts=20" );*/


if (Use["BDT"]) // Adaptive Boost
factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDT",
"!H:!V:NTrees=850:MinNodeSize=2.5%:MaxDepth=3:BoostType=AdaBoost:
AdaBoostBeta=0.5:UseBaggedBoost:BaggedSampleFraction=0.5:
SeparationType=GiniIndex:nCuts=20" );


if (Use["BDTB"]) // Bagging
factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDTB",
"!H:!V:NTrees=400:BoostType=Bagging:SeparationType=GiniIndex:nCuts=20" );

if (Use["BDTD"]) // Decorrelation + Adaptive Boost
factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDTD",
"!H:!V:NTrees=400:MinNodeSize=5%:MaxDepth=3:BoostType=AdaBoost:
SeparationType=GiniIndex:nCuts=20:VarTransform=Decorrelate" );

if (Use["BDTF"]) // Allow Using Fisher discriminant in node splitting for (strong) linearly
    correlated variables
factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDTF",
"!H:!V:NTrees=50:MinNodeSize=2.5%:UseFisherCuts:MaxDepth=3:
BoostType=AdaBoost:AdaBoostBeta=0.5:SeparationType=GiniIndex:nCuts=20" );

// RuleFit -- TMVA implementation of Friedman's method
if (Use["RuleFit"])
factory->BookMethod( dataloader, TMVA::Types::kRuleFit, "RuleFit",
"H:!V:RuleFitModule=RFTMVA:Model=ModRuleLinear:MinImp=0.001:
RuleMinDist=0.001:NTrees=20:fEventsMin=0.01:fEventsMax=0.5:
GDTau=-1.0:GDTauPrec=0.01:GDStep=0.01:GDNSteps=10000:GDErrScale=1.02" );

// For an example of the category classifier usage, see: runCategory
//
// --------------------------------------------------------------
//  Now you can optimize the setting (configuration) of the MVAs using the set of training
    events
// STILL EXPERIMENTAL and only implemented for BDT's !
//
//      factory->OptimizeAllMethods("SigEffAt001","Scan");
//      factory->OptimizeAllMethods("ROCIntegral","FitGA");
//
// --------------------------------------------------------------

// Now you can tell the factory to train, test, and evaluate the MVAs
//
// Train MVAs using the set of training events
factory->TrainAllMethods();

// Evaluate all MVAs using the set of test events
```

```
factory->TestAllMethods();

// Evaluate and compare performance of all configured MVAs
factory->EvaluateAllMethods();

// --------------------------------------------------------

// Save the output
outputFile->Close();

std::cout << "==> Wrote root file: " << outputFile->GetName() << std::endl;
std::cout << "==> RRQtrain is done!" << std::endl;

delete factory;
delete dataloader;
// Launch the GUI for the root macros
if (!gROOT->IsBatch()) TMVA::TMVAGui( outfileName );

return 0;
}

int main( int argc, char** argv )
{
// Select methods (don't look at this code - not of interest)
TString methodList;
for (int i=1; i<argc; i++) {
TString regMethod(argv[i]);
if(regMethod=="-b" || regMethod=="--batch") continue;
if (!methodList.IsNull()) methodList += TString(",");
methodList += regMethod;
}
return RRQtrain(methodList);
}
```

8.4.8 ML application ER and NR.

```
/// \ file
/// \ingroup tutorial_tmva
/// \notebook -nodraw
/// This macro provides a simple example on how to use the trained  classifiers
/// within an analysis  module
/// - Project   : TMVA - a Root-integrated toolkit for multivariate data analysis
/// - Package   : TMVA
/// - Exectuable: applicationRRQ
///
/// \macro_output
/// \macro_code
/// \author Andreas Hoecker

//    root -l ./applicationRRQ.C\(\"BDT\"\)


#include <cstdlib>
#include <vector>
#include <iostream>
#include <map>
#include <string>

#include "TFile.h"
#include "TTree.h"
#include "TString.h"
#include "TSystem.h"
#include "TROOT.h"
#include "TTree.h"
#include "TStopwatch.h"
#include "TMVA/Tools.h"
#include "TMVA/Reader.h"
#include "TMVA/MethodCuts.h"

using namespace TMVA;

void applicationRRQ( TString myMethodList = "" )
{

//---------------------------------------------------
// This loads the  library
TMVA::Tools::Instance();

// Default MVA methods to be trained + tested
std :: map<std::string,int> Use;

// Cut optimisation
Use["Cuts"]            = 1;
Use["CutsD"]           = 1;
```

```
Use["CutsPCA"]         = 0;
Use["CutsGA"]          = 0;
Use["CutsSA"]          = 0;
//
// 1-dimensional likelihood ("naive Bayes estimator")
Use["Likelihood"]      = 1;
Use["LikelihoodD"]     = 0; // the "D" extension indicates decorrelated input variables (see
    option strings)
Use["LikelihoodPCA"]   = 1; // the "PCA" extension indicates PCA-transformed input variables
    (see option strings)
Use["LikelihoodKDE"]   = 0;
Use["LikelihoodMIX"]   = 0;
//
// Mutidimensional likelihood and Nearest-Neighbour methods
Use["PDERS"]           = 1;
Use["PDERSD"]          = 0;
Use["PDERSPCA"]        = 0;
Use["PDEFoam"]         = 1;
Use["PDEFoamBoost"]    = 0; // uses generalised MVA method boosting
Use["KNN"]             = 1; // k-nearest neighbour method
//
// Linear Discriminant Analysis
Use["LD"]              = 1; // Linear Discriminant identical to Fisher
Use["Fisher"]          = 0;
Use["FisherG"]         = 0;
Use["BoostedFisher"]   = 0; // uses generalised MVA method boosting
Use["HMatrix"]         = 0;
//
// Function Discriminant analysis
Use["FDA_GA"]          = 1; // minimisation of user-defined function using Genetics Algorithm
Use["FDA_SA"]          = 0;
Use["FDA_MC"]          = 0;
Use["FDA_MT"]          = 0;
Use["FDA_GAMT"]        = 0;
Use["FDA_MCMT"]        = 0;
//
// Neural Networks (all are feed-forward Multilayer Perceptrons)
Use["MLP"]             = 0; // Recommended ANN
Use["MLPBFGS"]         = 0; // Recommended ANN with optional training method
Use["MLPBNN"]          = 1; // Recommended ANN with BFGS training method and bayesian
    regulator
Use["CFMlpANN"]        = 0; // Depreciated ANN from ALEPH
Use["TMlpANN"]         = 0; // ROOT's own ANN
Use["DNN_CPU"] = 0;         // CUDA-accelerated DNN training.
Use["DNN_GPU"] = 0;         // Multi-core accelerated DNN.
//
// Support Vector Machine
Use["SVM"]             = 1;
//
// Boosted Decision Trees
Use["BDT"]             = 1; // uses Adaptive Boost
```

```
Use["BDTG"]            = 0; // uses Gradient Boost
Use["BDTB"]            = 0; // uses Bagging
Use["BDTD"]            = 0; // decorrelation + Adaptive Boost
Use["BDTF"]            = 0; // allow usage of fisher discriminant for node splitting
//
// Friedman's RuleFit method, ie, an optimised series of cuts ("rules")
Use["RuleFit"]         = 1;
// ---------------------------------------
Use["Plugin"]          = 0;
Use["Category"]        = 0;
Use["SVM_Gauss"]       = 0;
Use["SVM_Poly"]        = 0;
Use["SVM_Lin"]         = 0;

std::cout << std::endl;
std::cout << "==> Start applicationRRQ" << std::endl;

// Select methods (don't look at this code - not of interest)
if (myMethodList != "") {
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++)
 it->second = 0;

std::vector<TString> mlist = gTools().SplitString( myMethodList, ',' );
for (UInt_t i=0; i<mlist.size(); i++) {
std::string regMethod(mlist[i]);

if (Use.find(regMethod) == Use.end()) {
std::cout << "Method \"" << regMethod
<< "\" not known in TMVA under this name. Choose among the following:" << std::endl;
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) {
std::cout << it->first << " ";
}
std::cout << std::endl;
return;
}
Use[regMethod] = 1;
}
}

// ---------------------------------------




// Create the Reader object

TMVA::Reader *reader = new TMVA::Reader( "Color:!Silent" );

// Create a set of variables and declare them to the reader
// - the variable names MUST corresponds in name and type to those given in the weight file(s)
     used
```

```
/*Float_t var1, var2;
Float_t var3, var4;
reader->AddVariable( "myvar1 := var1+var2", &var1 );
reader->AddVariable( "myvar2 := var1-var2", &var2 );
reader->AddVariable( "var3",                &var3 );
reader->AddVariable( "var4",                &var4 );*/

float precoiltNF,ytNF, qzpartOF,qrpart1OF,pzpartOF,prpartOF,qsummaxOF;
float PTNFchisq,ptNF,PTOFchisq,PTglitch1OFchisq,PTlfnoise1OFchisq,
QS1OFchisq,qsum1OF,QS2OFchisq,qsum2OF,qo1OF,qo2OF,qi1OF,qi2OF,qimean;


reader->AddVariable( "precoiltNF",&precoiltNF);

reader->AddVariable( "ytNF", &ytNF );
reader->AddVariable( "qzpartOF", &qzpartOF );
reader->AddVariable( "qrpart1OF", &qrpart1OF );
reader->AddVariable( "pzpartOF", &pzpartOF );
reader->AddVariable( "prpartOF", &prpartOF );
reader->AddVariable( "qsummaxOF", &qsummaxOF );

//reader->AddVariable( "DCAdiff:=abs(dcad1-dcad2)", &dcad1 );

// Spectator variables declared in the training have to be added to the reader, too
/*  Float_t spec1,spec2;
reader->AddSpectator( "spec1 := var1*2", &spec1 );
reader->AddSpectator( "spec2 := var1*3", &spec2 ); */



double cutvalue=-0.0528; //like
cout<<"THE CUT VALUE APPLIED IS============="<<cutvalue<<endl;



/* Float_t Category_cat1, Category_cat2, Category_cat3;
if (Use["Category"]){
// Add artificial spectators for distinguishing categories
reader->AddSpectator( "Category_cat1 := var3<=0",          &Category_cat1 );
reader->AddSpectator( "Category_cat2 := (var3>0)&&(var4<0)", &Category_cat2 );
reader->AddSpectator( "Category_cat3 := (var3>0)&&(var4>=0)", &Category_cat3 );
}
*/

// Book the MVA methods
//  Path :
     /home/viraj/Programs/root/tutorials/tmva/dataset/weights/Kstarweights/4featuresbdt
     //weight file name:  Kstarrun_BDT.weights.xml
//TString dir    = "/home/viraj/Programs/root/tutorials/tmva/";
TString dir    = "dataset/weights/"; //"/home/viraj/Programs/root/tutorials/tmva/";
```

```
TString prefix = "RRQtrain";

// Book method(s)
for (std::map<std::string,int>::iterator it = Use.begin(); it != Use.end(); it++) {
if (it->second) {
TString methodName = TString(it->first) + TString(" method");
TString weightfile = dir + prefix + TString("_") + TString(it->first) +
    TString(".weights.xml");
reader->BookMVA( methodName, weightfile );
}
}

// Book output histograms
UInt_t nbin = 100;
TH1F *histLk(0);
TH1F *histLkD(0);
TH1F *histLkPCA(0);
TH1F *histLkKDE(0);
TH1F *histLkMIX(0);
TH1F *histPD(0);
TH1F *histPDD(0);
TH1F *histPDPCA(0);
TH1F *histPDEFoam(0);
TH1F *histPDEFoamErr(0);
TH1F *histPDEFoamSig(0);
TH1F *histKNN(0);
TH1F *histHm(0);
TH1F *histFi(0);
TH1F *histFiG(0);
TH1F *histFiB(0);
TH1F *histLD(0);
TH1F *histNn(0);
TH1F *histNnbfgs(0);
TH1F *histNnbnn(0);
TH1F *histNnC(0);
TH1F *histNnT(0);
TH1F *histBdt(0);
TH1F *histBdtG(0);
TH1F *histBdtB(0);
TH1F *histBdtD(0);
TH1F *histBdtF(0);
TH1F *histRf(0);
TH1F *histSVMG(0);
TH1F *histSVMP(0);
TH1F *histSVML(0);
TH1F *histFDAMT(0);
TH1F *histFDAGA(0);
TH1F *histCat(0);
TH1F *histPBdt(0);
TH1F *histDnnGpu(0);
```

```
TH1F *histDnnCpu(0);

TCanvas *can;
// **********************Mass histogram


double precoiltNFrange_1=0;
double precoiltNFrange_2=170;
int  precoiltNFbinsize=450;
int  qbinsize=500; // Prev: 500, 160
double q_y_range_1=0; //Prev: -10, -2
double q_y_range_2=160; //Prev: 80, 14

TH2D *hERNR = new TH2D("hERNR","ER NR", precoiltNFbinsize, precoiltNFrange_1,
    precoiltNFrange_2, qbinsize, q_y_range_1, q_y_range_2);
hERNR->SetXTitle("precoiltNF [keV]");
hERNR->SetYTitle("qimean");
hERNR->GetYaxis()->SetTitleOffset(1.5);
hERNR->GetYaxis()->SetLabelSize(0.03);
hERNR->GetYaxis()->SetTitleSize(0.03);
//hERNR->SetLineColor(kGray+3);
//hERNR->SetFillColor(kGray+3);
//hERNR->SetMarkerColor(kGray+3);



TH2D *hERNR_ML = new TH2D("hERNR_ML","ER NR ML-classification",
    precoiltNFbinsize, precoiltNFrange_1, precoiltNFrange_2, qbinsize, q_y_range_1, q_y_range_2);
hERNR_ML->SetXTitle("precoiltNF [keV]");
hERNR_ML->SetYTitle("qimean");
hERNR_ML->GetYaxis()->SetTitleOffset(1.5);
hERNR_ML->GetYaxis()->SetLabelSize(0.03);
hERNR_ML->GetYaxis()->SetTitleSize(0.03);
//hERNR_ML->SetLineColor(kGray+3);
//hERNR_ML->SetFillColor(kGray+3);
//hERNR_ML->SetMarkerColor(kGray+3);


TH2D *hyieldE = new TH2D("hyieldE","Yield vs Recoil Energy", precoiltNFbinsize,
    precoiltNFrange_1, precoiltNFrange_2, 100, 0, 2);
hyieldE->SetXTitle("precoiltNF [keV]");
hyieldE->SetYTitle("Yield");
hyieldE->GetYaxis()->SetTitleOffset(1.5);
hyieldE->GetYaxis()->SetLabelSize(0.03);
hyieldE->GetYaxis()->SetTitleSize(0.03);


TH2D *hyieldE_ML = new TH2D("hyieldE_ML","Yield vs Recoil Energy", precoiltNFbinsize,
    precoiltNFrange_1, precoiltNFrange_2, 100, 0, 2);
hyieldE_ML->SetXTitle("precoiltNF [keV]");
hyieldE_ML->SetYTitle("Yield");
```

```
hyieldE_ML->GetYaxis()->SetTitleOffset(1.5);
hyieldE_ML->GetYaxis()->SetLabelSize(0.03);
hyieldE_ML->GetYaxis()->SetTitleSize(0.03);
```

```
if (Use["Likelihood"])     histLk     = new TH1F( "MVA_Likelihood", "MVA_Likelihood",
    nbin, -1, 1 );
if (Use["LikelihoodD"])    histLkD    = new TH1F( "MVA_LikelihoodD", "MVA_LikelihoodD",
    nbin, -1, 0.9999 );
if (Use["LikelihoodPCA"]) histLkPCA = new TH1F( "MVA_LikelihoodPCA",
    "MVA_LikelihoodPCA", nbin, -1, 1 );
if (Use["LikelihoodKDE"]) histLkKDE = new TH1F( "MVA_LikelihoodKDE",
    "MVA_LikelihoodKDE", nbin, -0.00001, 0.99999 );
if (Use["LikelihoodMIX"]) histLkMIX = new TH1F( "MVA_LikelihoodMIX",
    "MVA_LikelihoodMIX", nbin, 0, 1 );
if (Use["PDERS"])          histPD     = new TH1F( "MVA_PDERS",   "MVA_PDERS",
    nbin,  0, 1 );
if (Use["PDERSD"])         histPDD    = new TH1F( "MVA_PDERSD", "MVA_PDERSD",
    nbin,  0, 1 );
if (Use["PDERSPCA"])       histPDPCA = new TH1F( "MVA_PDERSPCA",
    "MVA_PDERSPCA", nbin, 0, 1 );
if (Use["KNN"])            histKNN    = new TH1F( "MVA_KNN",     "MVA_KNN",
    nbin,  0, 1 );
if (Use["HMatrix"])        histHm     = new TH1F( "MVA_HMatrix", "MVA_HMatrix",
    nbin, -0.95, 1.55 );
if (Use["Fisher"])         histFi     = new TH1F( "MVA_Fisher",  "MVA_Fisher",
    nbin, -4, 4 );
if (Use["FisherG"])        histFiG    = new TH1F( "MVA_FisherG", "MVA_FisherG",
    nbin, -1, 1 );
if (Use["BoostedFisher"]) histFiB    = new TH1F( "MVA_BoostedFisher",
    "MVA_BoostedFisher", nbin, -2, 2 );
if (Use["LD"])             histLD     = new TH1F( "MVA_LD",      "MVA_LD",
    nbin, -2, 2 );
if (Use["MLP"])            histNn     = new TH1F( "MVA_MLP",     "MVA_MLP",
    nbin, -1.25, 1.5 );
if (Use["MLPBFGS"])        histNnbfgs = new TH1F( "MVA_MLPBFGS", "MVA_MLPBFGS",
    nbin, -1.25, 1.5 );
if (Use["MLPBNN"])         histNnbnn  = new TH1F( "MVA_MLPBNN", "MVA_MLPBNN",
    nbin, -1.25, 1.5 );
if (Use["CFMlpANN"])       histNnC    = new TH1F( "MVA_CFMlpANN", "MVA_CFMlpANN",
    nbin, 0, 1 );
if (Use["TMlpANN"])        histNnT    = new TH1F( "MVA_TMlpANN", "MVA_TMlpANN",
    nbin, -1.3, 1.3 );
if (Use["DNN_GPU"]) histDnnGpu = new TH1F("MVA_DNN_GPU", "MVA_DNN_GPU", nbin,
    -0.1, 1.1);
if (Use["DNN_CPU"]) histDnnCpu = new TH1F("MVA_DNN_CPU", "MVA_DNN_CPU", nbin,
    -0.1, 1.1);
if (Use["BDT"])            histBdt    = new TH1F( "MVA_BDT",     "MVA_BDT",
    nbin, -0.8, 0.8 );
```

```
if (Use["BDTG"])          histBdtG    = new TH1F( "MVA_BDTG",    "MVA_BDTG",
    nbin, -1.0, 1.0 );
if (Use["BDTB"])          histBdtB    = new TH1F( "MVA_BDTB",    "MVA_BDTB",
    nbin, -1.0, 1.0 );
if (Use["BDTD"])          histBdtD    = new TH1F( "MVA_BDTD",    "MVA_BDTD",
    nbin, -0.8, 0.8 );
if (Use["BDTF"])          histBdtF    = new TH1F( "MVA_BDTF",    "MVA_BDTF",
    nbin, -1.0, 1.0 );
if (Use["RuleFit"])       histRf      = new TH1F( "MVA_RuleFit", "MVA_RuleFit",
    nbin, -2.0, 2.0 );
if (Use["SVM_Gauss"])     histSVMG    = new TH1F( "MVA_SVM_Gauss", "MVA_SVM_Gauss",
    nbin, 0.0, 1.0 );
if (Use["SVM_Poly"])      histSVMP    = new TH1F( "MVA_SVM_Poly", "MVA_SVM_Poly",
    nbin,  0.0, 1.0 );
if (Use["SVM_Lin"])       histSVML    = new TH1F( "MVA_SVM_Lin", "MVA_SVM_Lin",
    nbin,  0.0, 1.0 );
if (Use["FDA_MT"])        histFDAMT = new TH1F( "MVA_FDA_MT", "MVA_FDA_MT",
    nbin, -2.0, 3.0 );
if (Use["FDA_GA"])        histFDAGA = new TH1F( "MVA_FDA_GA", "MVA_FDA_GA",
    nbin, -2.0, 3.0 );
if (Use["Category"])      histCat     = new TH1F( "MVA_Category", "MVA_Category",
    nbin, -2., 2. );
if (Use["Plugin"])        histPBdt    = new TH1F( "MVA_PBDT",    "MVA_BDT",
    nbin, -0.8, 0.8 );

// PDEFoam also returns per-event error, fill in histogram, and also fill significance
if (Use["PDEFoam"]) {
histPDEFoam  = new TH1F( "MVA_PDEFoam", "MVA_PDEFoam",          nbin,  0, 1 );
histPDEFoamErr = new TH1F( "MVA_PDEFoamErr", "MVA_PDEFoam error", nbin, 0, 1 );
histPDEFoamSig = new TH1F( "MVA_PDEFoamSig", "MVA_PDEFoam significance", nbin, 0,
    10 );
}

// Book example histogram for probability (the other methods are done similarly)
TH1F *probHistFi(0), *rarityHistFi(0);
if (Use["Fisher"]) {
probHistFi   = new TH1F( "MVA_Fisher_Proba", "MVA_Fisher_Proba", nbin, 0, 1 );
rarityHistFi = new TH1F( "MVA_Fisher_Rarity", "MVA_Fisher_Rarity", nbin, 0, 1 );
}

// Prepare input tree (this must be replaced by your data source)
// in this example, there is a toy tree with signal and one with background events
// we'll later on use only the "signal" events for the test in this example.
//
TFile *input(0);

//  **********************************************************CHANGE INPUT FILE
    NAME*************************************************************
// TString fname = "./RRQ_WS.root";
TString fname = "./RRQ_NR_cf.root";
```

```
//TString fname = "./likesignplusplusfeatures.root";
// TString fname = "./likesignplusplusfeatures_0.8pt1.2.root";


if (!gSystem->AccessPathName( fname )) {
input = TFile::Open( fname ); // check if file in local directory exists
}
else {
TFile::SetCacheFileDir(".");
input = TFile::Open("http://root.cern.ch/files/tmva_class_example.root", "CACHEREAD");
 // if not: download from ROOT server
}
if (!input) {
std::cout << "ERROR: could not open data file" << std::endl;
exit(1);
}
std::cout << "--- TMVAClassificationApp : Using input file: " <<
 input->GetName() << std::endl;

// Event loop

// Prepare the event tree
// - Here the variable names have to corresponds to your tree
// - You can use the same variables as above which is slightly faster,
//   but of course you can use different ones and copy the values inside the event loop
//
std::cout << "--- Select signal sample" << std::endl;
TTree* theTree = (TTree*)input->Get("tree_bkg");
// TTree* theTree = (TTree*)input->Get("tree_bkg");
// Float_t userVar1, userVar2;

/* theTree->SetBranchAddress( "var1", &userVar1 );
theTree->SetBranchAddress( "var2", &userVar2 );
theTree->SetBranchAddress( "var3", &var3 );
theTree->SetBranchAddress( "var4", &var4 );*/

//  Int_t Event,mult_s;

theTree->SetBranchAddress("PTNFchisq",&PTNFchisq);
theTree->SetBranchAddress("ptNF",&ptNF);
theTree->SetBranchAddress("PTOFchisq",&PTOFchisq);
theTree->SetBranchAddress("PTglitch1OFchisq",&PTglitch1OFchisq);
theTree->SetBranchAddress("PTlfnoise1OFchisq",&PTlfnoise1OFchisq);
theTree->SetBranchAddress("QS1OFchisq",&QS1OFchisq);
theTree->SetBranchAddress("qsum1OF",&qsum1OF);
theTree->SetBranchAddress("QS2OFchisq",&QS2OFchisq);
theTree->SetBranchAddress("qsum2OF",&qsum2OF);
theTree->SetBranchAddress("qo1OF",&qo1OF);
theTree->SetBranchAddress("qo2OF",&qo2OF);
theTree->SetBranchAddress("qi1OF",&qi1OF);
theTree->SetBranchAddress("qi2OF",&qi2OF);
```

```
theTree->SetBranchAddress("qimean",&qimean);


theTree->SetBranchAddress("precoiltNF",&precoiltNF);
theTree->SetBranchAddress("ytNF", &ytNF);
theTree->SetBranchAddress("qzpartOF", &qzpartOF);
theTree->SetBranchAddress("qrpart1OF", &qrpart1OF);
theTree->SetBranchAddress("pzpartOF", &pzpartOF);
theTree->SetBranchAddress("prpartOF", &prpartOF);
theTree->SetBranchAddress("qsummaxOF", &qsummaxOF);

// theTree->SetBranchAddress( "abs(dcad1-dcad2)", &userVar1 );
cout<<"The number of entries in
 theTree====================="<<theTree->GetEntries()<<endl;

// Efficiency calculator for cut method
Int_t      nSelCutsGA = 0;
Double_t effS         = 0.7;

std::vector<Float_t> vecVar(4); // vector for EvaluateMVA tests

std::cout << "--- Processing: " << theTree->GetEntries() << " events" << std::endl;
TStopwatch sw;
sw.Start();

//************************EVENT LOOP ************************



float  precoiltNF_ER,ytNF_ER,
    qzpartOF_ER,qrpart1OF_ER,pzpartOF_ER,prpartOF_ER,qsummaxOF_ER;
float  PTNFchisq_ER,ptNF_ER,PTOFchisq_ER,PTglitch1OFchisq_ER,
PTlfnoise1OFchisq_ER,
QS1OFchisq_ER,qsum1OF_ER,QS2OFchisq_ER,qsum2OF_ER,qo1OF_ER,qo2OF_ER,
qi1OF_ER,qi2OF_ER,qimean_ER;


TFile *fout = new TFile("A.root","RECREATE");
TTree *tree_ER = new TTree("tree_ER","tree_ER");
tree_ER->Branch("precoiltNF",&precoiltNF_ER,"precoiltNF_ER/F");
tree_ER->Branch("ytNF",&ytNF_ER,"ytNF_ER/F");
tree_ER->Branch("qzpartOF",&qzpartOF_ER,"qzpartOF_ER/F");
tree_ER->Branch("qrpart1OF",&qrpart1OF_ER,"qrpart1OF_ER/F");
tree_ER->Branch("pzpartOF",&pzpartOF_ER,"pzpartOF_ER/F");
tree_ER->Branch("prpartOF",&prpartOF_ER,"prpartOF_ER/F");
tree_ER->Branch("qsummaxOF",&qsummaxOF_ER,"qsummaxOF_ER/F");

tree_ER->Branch("PTNFchisq",&PTNFchisq_ER,"PTNFchisq_ER/F");
tree_ER->Branch("ptNF",&ptNF_ER,"ptNF_ER/F");
tree_ER->Branch("PTOFchisq",&PTOFchisq_ER,"PTOFchisq_ER/F");
```

```
tree_ER->Branch("PTglitch1OFchisq",&PTglitch1OFchisq_ER,"PTglitch1OFchisq_ER/F");
tree_ER->Branch("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_ER,"PTlfnoise1OFchisq_ER/F");
tree_ER->Branch("QS1OFchisq",&QS1OFchisq_ER,"QS1OFchisq_ER/F");
tree_ER->Branch("qsum1OF",&qsum1OF_ER,"qsum1OF_ER/F");
tree_ER->Branch("QS2OFchisq",&QS2OFchisq_ER,"QS2OFchisq_ER/F");
tree_ER->Branch("qsum2OF",&qsum2OF_ER,"qsum2OF_ER/F");
tree_ER->Branch("qo1OF",&qo1OF_ER,"qo1OF_ER/F");
tree_ER->Branch("qo2OF",&qo2OF_ER,"qo2OF_ER/F");
tree_ER->Branch("qi1OF",&qi1OF_ER,"qi1OF_ER/F");
tree_ER->Branch("qi2OF",&qi2OF_ER,"qi2OF_ER/F");
tree_ER->Branch("qimean",&qimean_ER,"qimean_ER/F");

float  precoiltNF_NR,ytNF_NR,
    qzpartOF_NR,qrpart1OF_NR,pzpartOF_NR,prpartOF_NR,qsummaxOF_NR;
float  PTNFchisq_NR,ptNF_NR,PTOFchisq_NR,PTglitch1OFchisq_NR,PTlfnoise1OFchisq_NR,
QS1OFchisq_NR,qsum1OF_NR,QS2OFchisq_NR,qsum2OF_NR,
qo1OF_NR,qo2OF_NR,qi1OF_NR,qi2OF_NR,qimean_NR;


TTree *tree_NR = new TTree("tree_NR","tree_NR");
tree_NR->Branch("precoiltNF",&precoiltNF_NR,"precoiltNF_NR/F");
tree_NR->Branch("ytNF",&ytNF_NR,"ytNF_NR/F");
tree_NR->Branch("qzpartOF",&qzpartOF_NR,"qzpartOF_NR/F");
tree_NR->Branch("qrpart1OF",&qrpart1OF_NR,"qrpart1OF_NR/F");
tree_NR->Branch("pzpartOF",&pzpartOF_NR,"pzpartOF_NR/F");
tree_NR->Branch("prpartOF",&prpartOF_NR,"prpartOF_NR/F");
tree_NR->Branch("qsummaxOF",&qsummaxOF_NR,"qsummaxOF_NR/F");

tree_NR->Branch("PTNFchisq",&PTNFchisq_NR,"PTNFchisq_NR/F");
tree_NR->Branch("ptNF",&ptNF_NR,"ptNF_NR/F");
tree_NR->Branch("PTOFchisq",&PTOFchisq_NR,"PTOFchisq_NR/F");
tree_NR->Branch("PTglitch1OFchisq",&PTglitch1OFchisq_NR,"PTglitch1OFchisq_NR/F");
tree_NR->Branch("PTlfnoise1OFchisq",&PTlfnoise1OFchisq_NR,"PTlfnoise1OFchisq_NR/F");
tree_NR->Branch("QS1OFchisq",&QS1OFchisq_NR,"QS1OFchisq_NR/F");
tree_NR->Branch("qsum1OF",&qsum1OF_NR,"qsum1OF_NR/F");
tree_NR->Branch("QS2OFchisq",&QS2OFchisq_NR,"QS2OFchisq_NR/F");
tree_NR->Branch("qsum2OF",&qsum2OF_NR,"qsum2OF_NR/F");
tree_NR->Branch("qo1OF",&qo1OF_NR,"qo1OF_NR/F");
tree_NR->Branch("qo2OF",&qo2OF_NR,"qo2OF_NR/F");
tree_NR->Branch("qi1OF",&qi1OF_NR,"qi1OF_NR/F");
tree_NR->Branch("qi2OF",&qi2OF_NR,"qi2OF_NR/F");
tree_NR->Branch("qimean",&qimean_NR,"qimean_NR/F");




for  (Long64_t ievt=0; ievt<theTree->GetEntries();ievt++)
{
// cout<<ievt<<endl;
if  (ievt%50000 == 0) std::cout << "--- ... Processing event: " << ievt << std::endl;
```

```
theTree->GetEntry(ievt);

//  theTree->GetEntry(k1);
//var1 = userVar1 + userVar2;
//var2 = userVar1 - userVar2;

// Return the MVA outputs and fill into histograms

if (Use["CutsGA"]) {
// Cuts is a special case: give the desired signal efficienciy
Bool_t passed = reader->EvaluateMVA( "CutsGA method", effS );
if (passed) nSelCutsGA++;
}

if (Use["Likelihood"    ])   histLk      ->Fill( reader->EvaluateMVA( "Likelihood method" )
    );
if (Use["LikelihoodD"   ])   histLkD     ->Fill( reader->EvaluateMVA( "LikelihoodD method"
    ) );
if (Use["LikelihoodPCA"])  histLkPCA  ->Fill( reader->EvaluateMVA( "LikelihoodPCA
    method" ) );
if (Use["LikelihoodKDE"])  histLkKDE  ->Fill( reader->EvaluateMVA( "LikelihoodKDE
    method" ) );
if (Use["LikelihoodMIX"])  histLkMIX  ->Fill( reader->EvaluateMVA( "LikelihoodMIX
    method" ) );
if (Use["PDERS"         ])   histPD      ->Fill( reader->EvaluateMVA( "PDERS method"  ) );
if (Use["PDERSD"        ])   histPDD     ->Fill( reader->EvaluateMVA( "PDERSD method" ) );
if (Use["PDERSPCA"      ])   histPDPCA   ->Fill( reader->EvaluateMVA( "PDERSPCA
    method" ) );
if (Use["KNN"           ])   histKNN     ->Fill( reader->EvaluateMVA( "KNN method"     ) );
if (Use["HMatrix"       ])   histHm      ->Fill( reader->EvaluateMVA( "HMatrix method" ) );
if (Use["Fisher"        ])   histFi      ->Fill( reader->EvaluateMVA( "Fisher method"  ) );
if (Use["FisherG"       ])   histFiG     ->Fill( reader->EvaluateMVA( "FisherG method" ) );
if (Use["BoostedFisher"])  histFiB     ->Fill( reader->EvaluateMVA( "BoostedFisher
    method" ) );
if (Use["LD"            ])   histLD      ->Fill( reader->EvaluateMVA( "LD method"      ) );
if (Use["MLP"           ])   histNn      ->Fill( reader->EvaluateMVA( "MLP method"     ) );
if (Use["MLPBFGS"       ])   histNnbfgs  ->Fill( reader->EvaluateMVA( "MLPBFGS method" )
    );
if (Use["MLPBNN"        ])   histNnbnn   ->Fill( reader->EvaluateMVA( "MLPBNN method" ) );
if (Use["CFMlpANN"      ])   histNnC     ->Fill( reader->EvaluateMVA( "CFMlpANN method"
    ) );
if (Use["TMlpANN"       ])   histNnT     ->Fill( reader->EvaluateMVA( "TMlpANN method" )
    );
if (Use["DNN_GPU"]) histDnnGpu->Fill(reader->EvaluateMVA("DNN_GPU method"));
if (Use["DNN_CPU"]) histDnnCpu->Fill(reader->EvaluateMVA("DNN_CPU method"));


if (Use["BDT"           ])     // ********************BDT*******************************
{
if (qimean<1) continue;
```

```
histBdt     ->Fill( reader->EvaluateMVA( "BDT method"     ) );
if (reader->EvaluateMVA( "BDT method")>cutvalue)
{
hERNR_ML->Fill(precoiltNF,qimean);
hyieldE_ML->Fill(precoiltNF,ytNF);

precoiltNF_NR=precoiltNF;
ytNF_NR=ytNF;
qzpartOF_NR=qzpartOF;
qrpart1OF_NR=qrpart1OF;
pzpartOF_NR=pzpartOF;
prpartOF_NR=prpartOF;
qsummaxOF_NR=qsummaxOF;

PTNFchisq_NR=PTNFchisq;
ptNF_NR=ptNF;
PTOFchisq_NR=PTOFchisq;
PTglitch1OFchisq_NR=PTglitch1OFchisq;
PTlfnoise1OFchisq_NR=PTlfnoise1OFchisq;
QS1OFchisq_NR=QS1OFchisq;
qsum1OF_NR=qsum1OF;
QS2OFchisq_NR=QS2OFchisq;
qsum2OF_NR=qsum2OF;
qo1OF_NR=qo1OF;
qo2OF_NR=qo2OF;
qi1OF_NR=qi1OF;
qi2OF_NR=qi2OF;
qimean_NR=0.5*(qi1OF+qi2OF);

tree_NR->Fill();


}
//   hERNR->Fill(precoiltNF,qimean);
if (reader->EvaluateMVA( "BDT method")<cutvalue)
{
hERNR->Fill(precoiltNF,qimean);
hyieldE->Fill(precoiltNF,ytNF);


precoiltNF_ER=precoiltNF;
ytNF_ER=ytNF;
qzpartOF_ER=qzpartOF;
qrpart1OF_ER=qrpart1OF;
pzpartOF_ER=pzpartOF;
prpartOF_ER=prpartOF;
qsummaxOF_ER=qsummaxOF;

PTNFchisq_ER=PTNFchisq;
ptNF_ER=ptNF;
PTOFchisq_ER=PTOFchisq;
```

```
PTglitch1OFchisq_ER=PTglitch1OFchisq;
PTlfnoise1OFchisq_ER=PTlfnoise1OFchisq;
QS1OFchisq_ER=QS1OFchisq;
qsum1OF_ER=qsum1OF;
QS2OFchisq_ER=QS2OFchisq;
qsum2OF_ER=qsum2OF;
qo1OF_ER=qo1OF;
qo2OF_ER=qo2OF;
qi1OF_ER=qi1OF;
qi2OF_ER=qi2OF;
qimean_ER=0.5*(qi1OF+qi2OF);

tree_ER->Fill();



}
//  cout<< reader->EvaluateMVA( "BDT method"    )<<
//"  ----------------------------------------"<<invmass<<endl;

}
```

```
if (Use["BDTG"          ])    histBdtG   ->Fill( reader->EvaluateMVA( "BDTG method"   ) );
if (Use["BDTB"          ])    histBdtB   ->Fill( reader->EvaluateMVA( "BDTB method"   ) );
if (Use["BDTD"          ])    histBdtD   ->Fill( reader->EvaluateMVA( "BDTD method"   ) );
if (Use["BDTF"          ])    histBdtF   ->Fill( reader->EvaluateMVA( "BDTF method"   ) );
if (Use["RuleFit"       ])    histRf     ->Fill( reader->EvaluateMVA( "RuleFit method" ) );
if (Use["SVM_Gauss"     ])    histSVMG ->Fill( reader->EvaluateMVA( "SVM_Gauss method" )
    );
if (Use["SVM_Poly"      ])    histSVMP ->Fill( reader->EvaluateMVA( "SVM_Poly method" ) );
if (Use["SVM_Lin"       ])    histSVML ->Fill( reader->EvaluateMVA( "SVM_Lin method" ) );
if (Use["FDA_MT"        ])    histFDAMT ->Fill( reader->EvaluateMVA( "FDA_MT method" )
    );
if (Use["FDA_GA"        ])    histFDAGA ->Fill( reader->EvaluateMVA( "FDA_GA method" )
    );
if (Use["Category"      ])    histCat    ->Fill( reader->EvaluateMVA( "Category method" ) );
if (Use["Plugin"        ])    histPBdt   ->Fill( reader->EvaluateMVA( "P_BDT method"   ) );


//cout<< reader->EvaluateMVA( "BDT method"    )<<"
    ----------------------------------------"<<costheta<<endl;
// Retrieve also per-event error
if (Use["PDEFoam"]) {
```

```
Double_t val = reader->EvaluateMVA( "PDEFoam method" );
Double_t err = reader->GetMVAError();
histPDEFoam ->Fill( val );
histPDEFoamErr->Fill( err );
if (err>1.e-50) histPDEFoamSig->Fill( val/err );
}

// Retrieve probability instead of MVA output
if (Use["Fisher"])    {
probHistFi  ->Fill( reader->GetProba ( "Fisher method" ) );
rarityHistFi->Fill( reader->GetRarity( "Fisher method" ) );
}

}//---------------------------------EVENT LOOP
    ENDS-----------------------//
can = new TCanvas("can","",10,10,600,600);
can->SetLeftMargin(0.2);
can->SetRightMargin(0.05);
can->SetTopMargin(0.05);
can->SetBottomMargin(0.13);
can->cd();
gStyle->SetOptStat(0);
hERNR_ML->GetXaxis()->SetTitle("precoiltNF (keV)");
hERNR_ML->GetXaxis()->SetTitleOffset(1.2);
hERNR_ML->GetYaxis()->SetTitleOffset(1.7);
hERNR_ML->GetYaxis()->SetTitle("qimean");
hERNR_ML->SetMarkerColor(3); //3=Green
hERNR_ML->SetMarkerStyle(15);
hERNR_ML->SetMarkerSize(1.0);
hERNR_ML->Draw();

hERNR->GetXaxis()->SetTitle("precoiltNF (keV)");
hERNR->GetXaxis()->SetTitleOffset(1.2);
hERNR->GetYaxis()->SetTitleOffset(1.7);
hERNR->GetYaxis()->SetTitle("qimean");
hERNR->SetMarkerColor(2); //2=Red
hERNR->SetMarkerStyle(15);
hERNR->SetMarkerSize(1.0);
hERNR->Draw("same");
TLine *line1 = new TLine(10,7,170,95);
//TLine *line1 = new TLine((3-c)/m,3,80,ylinefun(80));
//    line1->SetLineWidth(3);
// line1->SetLineColor(2);
//   line1->Draw("same");


TLegend *l1 = new TLegend(.5365772,0.7801394,.6708054,0.8797909);
l1->SetTextFont(42);
l1->SetBorderSize(0);
l1->SetFillStyle(0);
//l1->SetFillColor(0);
```

```
l1->SetMargin(0.25);
l1->SetTextSize(0.03);
l1->SetEntrySeparation(0.5);
l1->AddEntry(hERNR_ML,"NR Signal by ML ","lpf");
l1->AddEntry(hERNR,"ER Background","lpf");
l1->Draw("same");


can->SaveAs("h2d.jpg");

TCanvas *can2;
can2 = new TCanvas("can2","",10,10,600,600);
can2->SetLeftMargin(0.2);
can2->SetRightMargin(0.05);
can2->SetTopMargin(0.05);
can2->SetBottomMargin(0.13);
can2->cd();

hyieldE_ML->GetXaxis()->SetTitle("precoiltNF (keV)");
hyieldE_ML->GetXaxis()->SetTitleOffset(1.2);
hyieldE_ML->GetYaxis()->SetTitleOffset(1.7);
hyieldE_ML->GetYaxis()->SetTitle("yield");
hyieldE_ML->SetMarkerColor(3); //3=Green
hyieldE_ML->SetMarkerStyle(15);
hyieldE_ML->SetMarkerSize(1.0);
hyieldE_ML->Draw();

hyieldE->GetXaxis()->SetTitle("precoiltNF (keV)");
hyieldE->GetXaxis()->SetTitleOffset(1.2);
hyieldE->GetYaxis()->SetTitleOffset(1.7);
hyieldE->GetYaxis()->SetTitle("qimean");
hyieldE->SetMarkerColor(96); //2=Red
hyieldE->SetMarkerStyle(15);
hyieldE->SetMarkerSize(1.0);
hyieldE->Draw("same");


TLegend *l2 = new TLegend(.5365772,0.7801394,.6708054,0.8797909);
l2->SetTextFont(42);
l2->SetBorderSize(0);
l2->SetFillStyle(0);
//l1->SetFillColor(0);
l2->SetMargin(0.25);
l2->SetTextSize(0.03);
l2->SetEntrySeparation(0.5);
l2->AddEntry(hyieldE_ML,"NR Band by ML ","p");
l2->AddEntry(hyieldE,"ER Band by ML","p");
l2->Draw("same");
```

```
// Get elapsed time
sw.Stop();
std::cout << "--- End of event loop: "; sw.Print();

// Get efficiency for cuts classifier
if (Use["CutsGA"]) std::cout << "--- Efficiency for CutsGA method: " <<
    double(nSelCutsGA)/theTree->GetEntries()
<< " (for a required signal efficiency of " << effS << ")" << std::endl;

if (Use["CutsGA"]) {

// test: retrieve cuts for particular signal efficiency
// CINT ignores dynamic_casts so we have to use a cuts-secific Reader function to acces the
    pointer
TMVA::MethodCuts* mcuts = reader->FindCutsMVA( "CutsGA method" ) ;

if (mcuts) {
std::vector<Double_t> cutsMin;
std::vector<Double_t> cutsMax;
mcuts->GetCuts( 0.7, cutsMin, cutsMax );
std::cout << "--- ------------------------------" << std::endl;
std::cout << "--- Retrieve cut values for signal efficiency of 0.7 from Reader" << std::endl;
for (UInt_t ivar=0; ivar<cutsMin.size(); ivar++) {
std::cout << "... Cut: "
<< cutsMin[ivar]
<< " < \""
<< mcuts->GetInputVar(ivar)
<< "\" <= "
<< cutsMax[ivar] << std::endl;
}
std::cout << "--- ------------------------------" << std::endl;
}
}

// Write histograms

TFile *target  = new TFile( "TMVApp.root","RECREATE" );
if (Use["Likelihood"   ]) histLk      ->Write();
if (Use["LikelihoodD"  ]) histLkD     ->Write();
if (Use["LikelihoodPCA"]) histLkPCA   ->Write();
if (Use["LikelihoodKDE"]) histLkKDE   ->Write();
if (Use["LikelihoodMIX"]) histLkMIX   ->Write();
if (Use["PDERS"        ]) histPD      ->Write();
if (Use["PDERSD"       ]) histPDD     ->Write();
if (Use["PDERSPCA"     ]) histPDPCA   ->Write();
if (Use["KNN"          ]) histKNN     ->Write();
if (Use["HMatrix"      ]) histHm      ->Write();
if (Use["Fisher"       ]) histFi      ->Write();
if (Use["FisherG"      ]) histFiG     ->Write();
if (Use["BoostedFisher"]) histFiB     ->Write();
if (Use["LD"           ]) histLD      ->Write();
```

```
if (Use["MLP"          ])    histNn      ->Write();
if (Use["MLPBFGS"      ])    histNnbfgs  ->Write();
if (Use["MLPBNN"       ])    histNnbnn   ->Write();
if (Use["CFMlpANN"     ])    histNnC     ->Write();
if (Use["TMlpANN"      ])    histNnT     ->Write();
if (Use["DNN_GPU"]) histDnnGpu->Write();
if (Use["DNN_CPU"]) histDnnCpu->Write();

if (Use["BDT"          ])    {


histBdt     ->Write();
hERNR->Write();
hERNR_ML->Write();
hyieldE->Write();
hyieldE_ML->Write();

}


if (Use["BDTG"         ])    histBdtG    ->Write();
if (Use["BDTB"         ])    histBdtB    ->Write();
if (Use["BDTD"         ])    histBdtD    ->Write();
if (Use["BDTF"         ])    histBdtF    ->Write();
if (Use["RuleFit"      ])    histRf      ->Write();
if (Use["SVM_Gauss"    ])    histSVMG    ->Write();
if (Use["SVM_Poly"     ])    histSVMP    ->Write();
if (Use["SVM_Lin"      ])    histSVML    ->Write();
if (Use["FDA_MT"       ])    histFDAMT   ->Write();
if (Use["FDA_GA"       ])    histFDAGA   ->Write();
if (Use["Category"     ])    histCat     ->Write();
if (Use["Plugin"       ])    histPBdt    ->Write();

// Write also error and significance histos
if (Use["PDEFoam"]) { histPDEFoam->Write(); histPDEFoamErr->Write();
    histPDEFoamSig->Write(); }

// Write also probability hists
if (Use["Fisher"]) { if (probHistFi != 0) probHistFi->Write(); if (rarityHistFi != 0)
    rarityHistFi->Write(); }
target->Close();

std::cout << "--- Created root file: \"TMVApp.root\" containing the MVA output
    histograms" << std::endl;

delete reader;

std::cout << "==> applicationRRQ is done!" << std::endl << std::endl;

cout<<"The total number of
    entries----------------------------"<<theTree->GetEntries()<<endl;
```

```
cout<<" Number of entries in Background Invariant Mass
    histogram"<<setw(10)<<hERNR_ML->Integral(1,60)<<endl;
cout<<" Number of entries in Invariant Mass histogram
    "<<setw(10)<<hERNR->Integral(1,60)<<endl;
cout<<" Cut Value = "<<cutvalue<<endl;



fout->cd();
tree_ER->Write();
tree_NR->Write();
fout->Close();

}


int main( int argc, char** argv )
{
TString methodList;
for (int i=1; i<argc; i++) {
TString regMethod(argv[i]);
if(regMethod=="-b" || regMethod=="--batch") continue;
if (!methodList.IsNull()) methodList += TString(",");
methodList += regMethod;
}
applicationRRQ(methodList);
return 0;
}
```