INTERNAL NOTE SIN–R XX/2017

Boosted jet identification using Machine Learning techniques

R. M. Cobe, J. C. Ruiz Vargas, T. R. Tomei [1],
A. L. Polastro [2]

[1] *Núcleo de Computação Científica, UNESP - Univ Estadual Paulista*
[2] *Universisade Federal do ABC*

Contents

Figures

Tables

1. Introduction to Pythia 8

Responsible: Thiago, Amanda

Pythia 8 is a tool for generation of high-energy collisions. The particles are produced in vacuum. Pythia 8 contains many libraries of hard interactions and models for initial and final state parton showers showers, multiple parton-parton interactions, beam remnants, string fragmentation and particle decays It also has a set of utilities and interfaces to external programs. Thus, it is possible to run Pythia with applications such as *Root* or *Fastjet*.

Currently, the program only works with pp, $p\overline{p}$, e^-e^+ and $\mu^+\mu^-$ incoming beams. The list of all processes already implemented can be seen in [**?**].

2. Program Flow

In Pythia context, an event represents a collision (the main one). Thus, a collision between p^- and p^+ can be understood as an Pythia event.

Basically, a Pythia simulation can be done in three steps:

1. **Initialisation:** here the main settings of the event (main collision) such are set up. Some of these settings are:
   - the energy of the initial beams at the LHC
   - the processes switched on

   Sometimes, we can choose the particles which will appear in the list according to some particle attribute such as the transverse momentum.
2. **Generation of individual events:** the *event loop* and conditions to perform analysis.
3. **Statistics:** generation of statistics and histograms about the event

3. A Hello World Program

In order to clarify the program flow, there is an example of a Pythia simulation. This example is based on the main01 example of Pythia 8 [**?**].

So:

- **Event:** We generate 100 events of a proton-proton collision, energy of 8000 TeV at LHC.
- **Processes:** all processes from the HardQCD group are enabled. In order to understand other kinds of processes, see: http://home.thep.lu.se/ torbjorn/pythia81html/Welcome.html.
- **Listing Condition:** all particles which $pT \geq 20$ GeV.
- **Statistics:** histogram showing the number of particles which are final and charged X number of events with these number of particles.

Therefore:

```
#include "Pythia8/Pythia.h" // Include Pythia headers.
using namespace Pythia8; // Let Pythia8:: be implicit.

int main() {

// Set up generation.
Pythia pythia; // Declare Pythia object
pythia.readString("Beams:eCM = 8000."); // 8 TeV CM energy.
pythia.readString("HardQCD = on"); // Switch on all HardQCD processes.

pythia.init();
Hist mult("charged multiplicity", 100, -0.5, 799.5); // Set up histogram

// Begin event loop. Generate event. Skip if error.
  for (int iEvent = 0; iEvent < 100; ++iEvent) {
    if (!pythia.next()) continue;

// Find number of all final charged particles and fill histogram.
    int nCharged = 0;

    for (int i = 0; i < pythia.event.size(); ++i)
      if (pythia.event[i].isFinal() && pythia.event[i].isCharged())
        ++nCharged;
       mult.fill( nCharged ); //Fill the histogram
      // End of event loop. Done :)
 }
    pythia.stat();
```

```
    return 0;
}
```

Notes:

1. If there are many initial settings, it is better setting up these parameters from an external file `pythia.readFile("main03.cmnd")` instead of using the `pythia.readString("")` instruction.
2. Pythia will list in the output file only the particles which belong to the first event.

4. The Output

The Pythia 8 output files from *main*0*n* examples are basically the list of particles produced in the first event. If any listing condition such as a minimum transverse momentum is applied, this list will contain only particles which satisfy the condition. The output file can also contain histograms.

In the list, each line represents a particle and each column shows an attribute of it. These attributes are:

- **id:** this code indicates the kind of the particle according to the PDG particle codes []. For example: code 2212 is assigned to the proton (p) and code -2212 is assigned to the antiproton $\overline{p}$.
- **status:** status code. The full set of codes provides information on where and why a given particle was produced. The key feature is that a particle is assigned a positive status code when it is created, which then is negated if later it branches into other particles. The mechanism of this branching can be inferred from the status code of the daughters. Thus, at any given stage of the event-generation process, the current final state consists of the particles with positive status code.
- **mothers/daughters:** shows the relation between particles. Naturally, if a particle generates another, the first is the mother of the second.
- **m:** the particle mass.
- px**:** component x of the transverse momentum.
- py**:** component y of the transverse momentum.

- *pz*: component z of the transverse momentum.
- **e:** the fourth component of the momentum.

  In order to learn more attributes, see [**?**]

5. Link to Other Programs

6. Sample Generation

Responsible: Thiago, Amanda

6.1 Introduction to Pythia 8

Pythia 8 is a tool for generation of high-energy collisions. The particles are produced in vacuum. Pythia 8 contains many libraries of hard interactions and models for initial and final state parton showers showers, multiple parton-parton interactions, beam remnants, string fragmentation and particle decays It also has a set of utilities and interfaces to external programs. Thus, it is possible to run Pythia with applications such as *Root* or *Fastjet*.

Currently, the program only works with pp, $p\overline{p}$, e^-e^+ and $\mu^+\mu^-$ incoming beams. The list of all processes already implemented can be seen in [**?**].

6.2 Program Flow

In Pythia context, an event represents a collision (the main one). Thus, a collision between p^- and p^+ can be understood as an Pythia event.

Basically, a Pythia simulation can be done in three steps:

1. **Initialisation:** here the main settings of the event (main collision) such are set up. Some of these settings are:
   - the energy of the initial beams at the LHC
   - the processes switched on

   Sometimes, we can choose the particles which will appear in the list according to some particle attribute such as the transverse momentum.
2. **Generation of individual events:** the *event loop* and conditions to perform analysis.
3. **Statistics:** generation of statistics and histograms about the event

6.3 A Hello World Program

In order to clarify the program flow, there is an example of a Pythia simulation. This example is based on the main01 example of Pythia 8 [**?**].

So:

- **Event:** We generate 100 events of a proton-proton collision, energy of 8000 TeV at LHC.
- **Processes:** all processes from the HardQCD group are enabled. In order to understand other kinds of processes, see: http://home.thep.lu.se/ torbjorn/pythia81html/Welcome.html.
- **Listing Condition:** all particles which $p_T \geq 20$ GeV.
- **Statistics:** histogram showing the number of particles which are final and charged X number of events with these number of particles.

Therefore:

```
#include "Pythia8/Pythia.h" // Include Pythia headers.
using namespace Pythia8; // Let Pythia8:: be implicit.

int main()
{

    // Set up generation.
    Pythia pythia; // Declare Pythia object
    pythia.readString("Beams:eCM = 8000."); // 8 TeV CM energy.
    pythia.readString("HardQCD = on"); // Switch on all HardQCD processes.

    pythia.init();
    Hist mult("charged mult", 100, -0.5, 799.5); // Set up histogram

    // Begin event loop. Generate event. Skip if error.
    for (int iEvent = 0; iEvent < 100; ++iEvent) {
        if (!pythia.next()) continue;

        // Find number of all final charged particles and fill histogram.
        int nCharged = 0;

        for (int i = 0; i < pythia.event.size(); ++i)
            if (pythia.event[i].isFinal() && pythia.event[i].isCharged())
                ++nCharged;
        mult.fill(nCharged);    //Fill the histogram
        // End of event loop. Done :)
    }
    pythia.stat();
    return 0;
}
```

Notes:

1. If there are many initial settings, it is better setting up these parameters from an external file `pythia.readFile("main03.cmnd")` instead of using the `pythia.readString("")` instruction.

2. Pythia will list in the output file only the particles which belong to the first event.

6.4 The Output

The Pythia 8 output files from *main*0*n* examples are basically the list of particles produced in the first event. If any listing condition such as a minimum transverse momentum is applied, this list will contain only particles which satisfy the condition. The output file can also contain histograms.

In the list, each line represents a particle and each column shows an attribute of it. These attributes are:

- **id:** this code indicates the kind of the particle according to the PDG particle codes []. For example: code 2212 is assigned to the proton (p) and code -2212 is assigned to the antiproton $\overline{p}$.
- **status:** status code. The full set of codes provides information on where and why a given particle was produced. The key feature is that a particle is assigned a positive status code when it is created, which then is negated if later it branches into other particles. The mechanism of this branching can be inferred from the status code of the daughters. Thus, at any given stage of the event-generation process, the current final state consists of the particles with positive status code.
- **mothers/daughters:** shows the relation between particles. Naturally, if a particle generates another, the first is the mother of the second.
- **m:** the particle mass.
- *px***:** component x of the transverse momentum.
- *py***:** component y of the transverse momentum.
- *pz***:** component z of the transverse momentum.
- **e:** the fourth component of the momentum.

In order to learn more attributes, see [**?**]

6.5 Link to Other Programs

7. Jet Finding and Preprocessing

Responsible: Thiago

After generating jet events with Pythia, we convert them to *jet images* in order to treat them with Machine Learning tools that are already configured for image recognition. In contemporary particle physics experiments, the particles that make up a hadronic jet are reconstructed individually through the signals they leave in the different subsystems that make up the detector, and any further analysis is made on those reconstructed particle candidates. In this study we take a simpler approach, working only with

We consider a very simplistic detector model, composed only of a *segmented calorimeter* - a detector made of individual cells that are capable of measuring the total energy deposited in them by particles of any kind, but that cannot distinguish the signal deposited by different particles in the same cell. The calorimeter follows roughly the geometry of the real systems present at both the ATLAS and CMS experiments: the calorimeter is segmented in both η and ϕ directions, having 63 bins that cover the range $[-\pi, \pi]$ in ϕ and 50 bins that cover the range $[-2.5, 2.5]$ in η, leading to a grand total of 3150 cells of size (0.1×0.1). For each event, we convert it to a "calorimeter representation" by looping over the list of visible particles and adding its transverse energy to the corresponding cell. Figure 1 shows the representation of a dijet (background) event in the simulated calorimeter. This step is equivalent to the creation of the digital image itself, where information about the individual photons that hit the pixels is lost but the total amount of energy that hit each pixel is available.

7.1 Jet Finding

In order to identify hadronic jets in the calorimeter, we use the Fastjet framework []. We model each calorimeter cell by a massless 4-vector where the (η, ϕ) coordinates are taken as the center of the cell and its transverse energy E_T is taken as the total transverse energy deposited in the cell. This step brings us to an event representation in the form of a list of 3150 4-vectors that can be input to Fastjet in order to find hadronic jets. We use the Cambridge-Aachen jet algorithm [], with a characteristic size $R = 1.2$, and consider only jets with p_T above 30 GeV. We consider only the leading jet in each event, and only if it has p_T in a given range; the full set of p_T intervals considered in this study is 250–300, 450–500, 600–650, 750–800, 950–1000, 1150–1200 and 1400–1450 GeV.

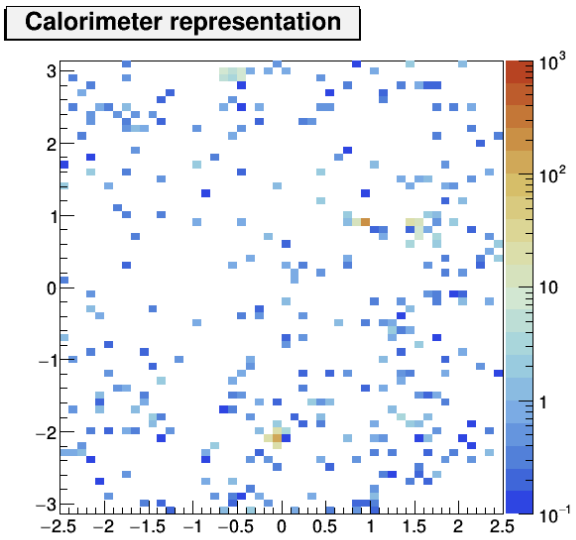

Figure 1: Example of calorimeter representation of event. The x-axis of the plot represents the η coordinate, while the y-axis represents the ϕ coordinate. The z-axis represents the total E_T in the each calorimeter cell. The event is a dijet (background) event, where the leading jet has p_T = 258 GeV, $\eta = 1.1$, $\phi = 0.88$ and mass = 69 GeV.

7.2 Preprocessing

Preprocessing must be done in the jet images in order to input them to the machine learning frameworks. The first step or preprocessing is *noise reduction*, which in our case can be done with the so called *jet trimming* []. Jet trimming is a particular technique for jet grooming that allows to reduce the effect of soft and collinear emissions that may spoil the jet kinematic resolution; it is also useful to minimise the effects of pileup interactions in the jet. We employ trimming with the k_T jet algorithm [] and a minimum subjet p_T fraction of 5%. In order to select good events for the list of inputs to the ML, we discard events where the trimmed jet has mass outside of the range 65–95 GeV. The trimmed jet has a reduced list of components; from this point on, we work only with the trimmed jet; operationally we achieve this by "zeroing" the cells that comprise the jet in the calorimeter and refilling that region with the filled components. That process is shown in Fig. 2

The next step in preprocessing is the definition of *points of interest* and the *alignment step*. The noise reduction step naturally defines the points of interest as the location of the subjets that emerge post-trimming. Operationally, this is done as follows:

- We locate the cell that contains the jet centroid, and save only the cells in a (25×25) rectangle around it. In this way we guarantee that all cells in a radius $R = 1.2$ are considered. We then do a translation to a local frame

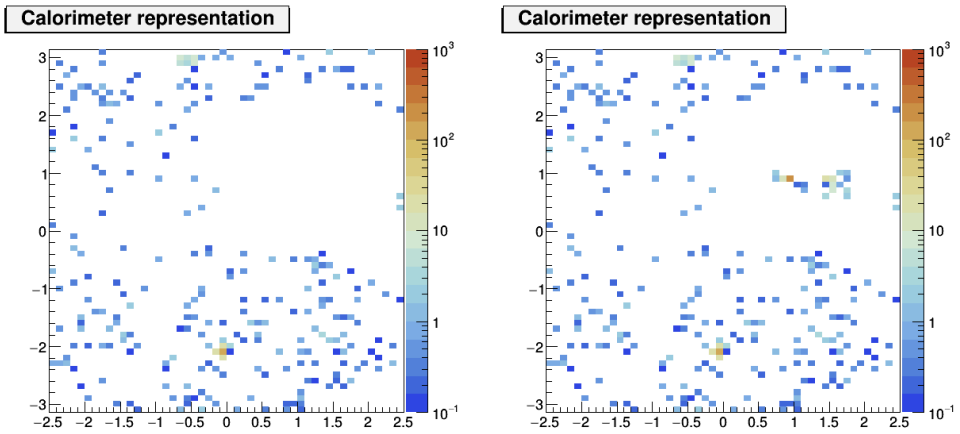

Figure 2: Example of calorimeter representation of event. The x-axis of the plot represents the η coordinate, while the y-axis represents the ϕ coordinate. The z-axis represents the total E_T in the each calorimeter cell. In the left plot, the cells located around the leading jet are zeroed out, while in the right plot those cells are replaced by the trimmed components.

(x, y) such that in this frame the centroid cell is at $(0, 0)$.

- We do a rotation to another local frame (x', y') such that in this frame the leading subjet has a higher y' coordinate than the subleading subjet, while their x' coordinates are identical; if there is only one subjet no rotation is performed.
- If the sum of E_T of the cells with $x' < 0$ is smaller than the sum of E_T of the cells with $x' \geq 0$, we do a reflection transformation $x'' = -x'$.

The final preprocessing step is the *normalisation step*. This step reduces the range of values in the features input to the ML. In our case, we simply rescale the E_T of the calorimeter cells by a constant factor c such that:

$$\frac{1}{c}\sum_i E_{T,i}^2 = 1 \tag{1}$$

The whole alignment and normalisation process is illustrated in Fig. 3

The content of the cells that comprise the jet is then saved in a plain text file. The contents of all the cells are saved sequentially, in the following format:

$$\begin{gathered} E_T(\eta_1, \phi_1), E_T(\eta_1, \phi_2), \ldots, E_T(\eta_1, \phi_{63}), \\ E_T(\eta_2, \phi_1), E_T(\eta_3, \phi_2), \ldots, E_T(\eta_2, \phi_{63}), \\ \ldots \\ E_T(\eta_{50}, \phi_1), E_T(\eta_{50}, \phi_2), \ldots, E_T(\eta_{50}, \phi_{63}), \end{gathered} \tag{2}$$

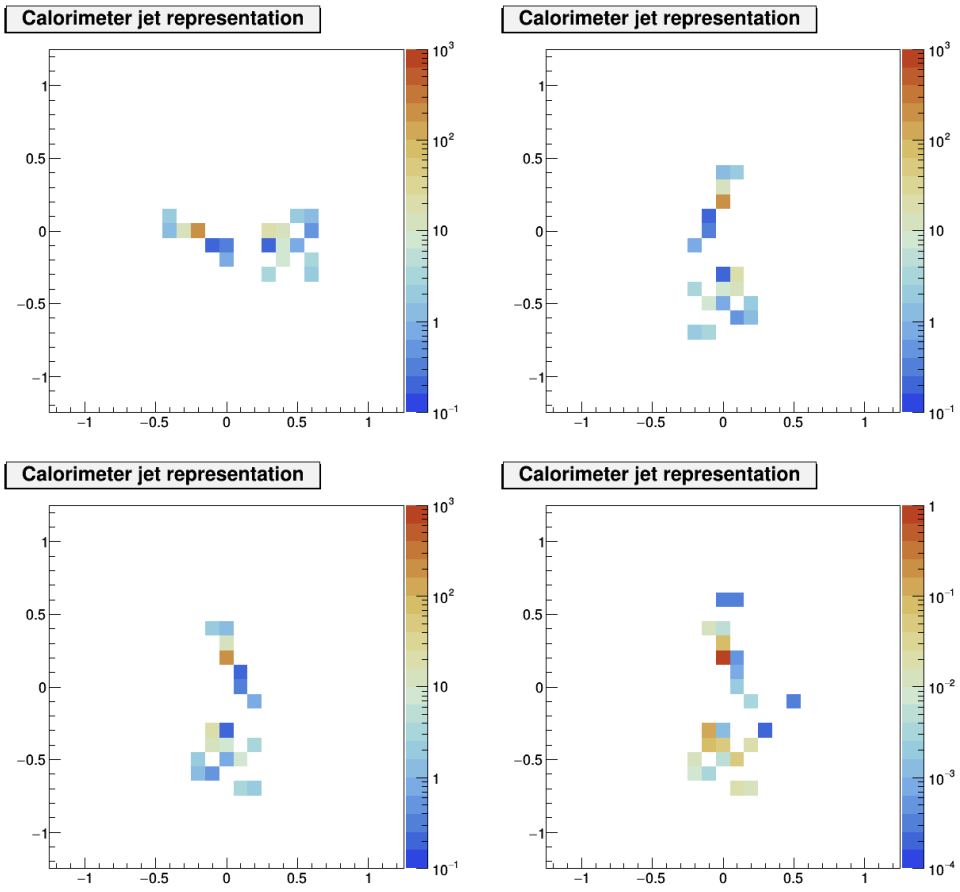

Figure 3: Example of calorimeter representation of jet. The x-axis of the plot represents the η coordinate, while the y-axis represents the ϕ coordinate. The z-axis represents the total E_T in the each calorimeter cell. In the top left plot, the coordinates have been changed such that the centroid cell is at $(0, 0)$. In the top right plot, a rotation has been executed such the vector that that connects the two subjets has no x component. In the bottom left plot, a reflection has been executed such that the sum E_T of all cells with $x < 0$ is smaller than the sum E_T of all cells with $x \geq 0$. Bottom right: all cells have their content rescaled by c such that of $\frac{1}{c}\sum_i E_{T,i}^2 = 1$.

8. Machine Learning Frameworks

Responsible: Jose, Rafael

8.1 Introduction to Machine Learning

Machine Learning (ML) is a broad field of research that provides general tools to solve complex problems. In this context, Artificial Neural Networks are inspired in the morphology and dynamics of the brain to create advanced computational models. The analogy remarks the ability of the human brain to perform classification tasks with outstanding performance; for instance, we are able to recognise images of familiar faces in less than 200 ms, while modern computers still require longer training sessions.

Fundamentally, Machine Learning is a mean of building a model of data. It can be understood as the development of mathematical models which helps understanding and recognizing patterns in data sets. These models have parameters which are fitted according to the observed data. Therefore, a data set works as the input for the model and once the model have been fit to previously seen data, it can be used to predict, recognize patterns, and even understand aspects of newly observed data. This way, the program can be considered to be learning from the data.

There are two kinds of Machine Learning: supervised and unsupervised learning. Supervised learning involves modelling a relationship between measured features of data and some label associated with it. Once the model is done, it can be used to apply labels to unknown data. An example of supervised learning consists of classifying jets into categories: signal or background.

Unsupervised learning involves modeling the features of a dataset without reference to any label. These models include tasks such as clustering and dimensionality reduction. Clustering algorithms identify distinct groups of data, while dimensionality reduction algorithms search for more succinct representations of the data. Unsupervised learning models use the intrinsic structure of the data to determine which subsets of data are related or not.

8.2 Machine Learning in High Energy Physics

The greatest challenge at the LHC at CERN is to collect and analyse data in a efficient way. Thus, sophisticated Machine Learning methods have been researched in order to tackle this problem.

In a collision between two protons p^+p^+ lots of particles are generated, as depicted in Fig. 4. Hadronic jets are the characteristic signature of quarks in high energy colliders. The identification of jets originating from boson decays and those coming from QCD processes is quite relevant in many analyses, in order to separate interesting signal events from unwanted backgrounds. Common methods for jet identification involve jet grooming techniques such as trimming, pruning, and soft drop, along with jet substructure variables such as N-subjettiness. The convenience of a jet classifier based on these features is justified by its simplicity and acceptable performance; nevertheless, more elaborated data analysis techniques may bring important improvements in terms of efficiency.

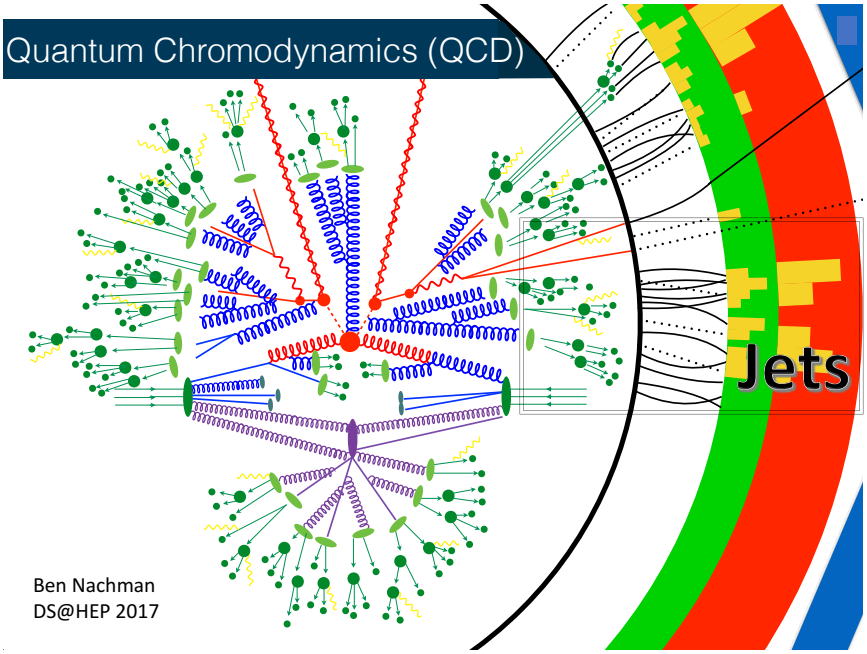

Figure 4: Quantum Chromodynamics (QCD) and formation of jets.

In this work, we handle the problem of jet classification in experimental high energy physics by using ML algorithms. First, we apply computer vision techniques to transform a hadronic jet into an image of 25×25 pixels, where each pixel represents a calorimetric cell (Fig. 5). Consequently, we evaluate the performance of different classifiers in a sample of simulated events; the signal corresponds to high

energy jets coming from W/Z events, and the background corresponds to similar jets coming from QCD processes. Finally, the results and benefits of the ML implementation are compared with the traditional N-subjettiness approach.

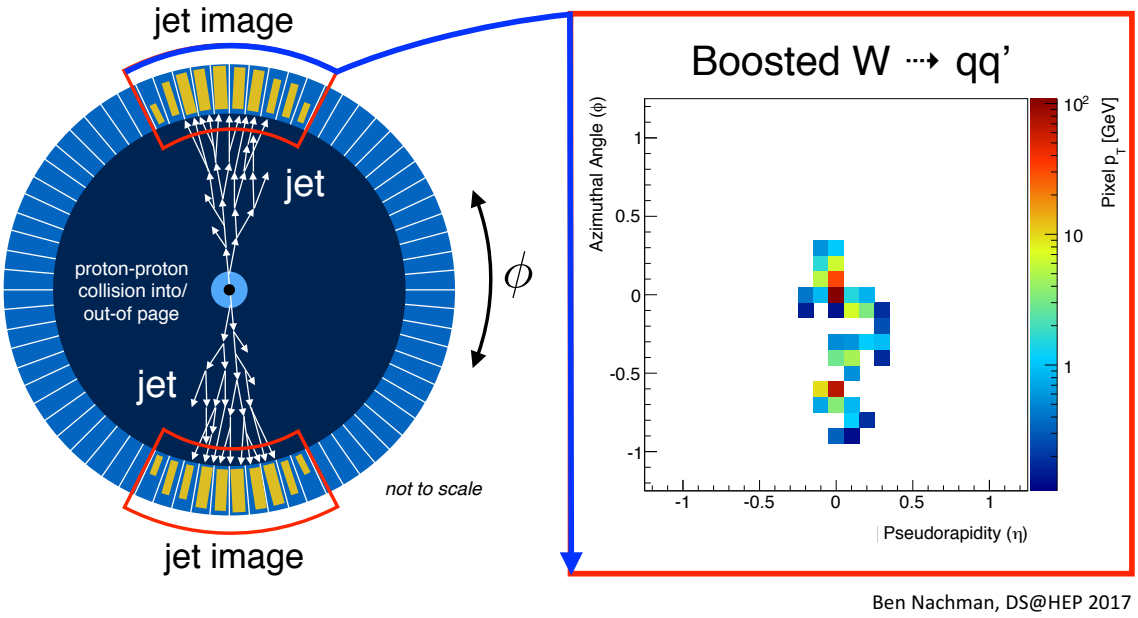

Figure 5: Image representation of a jet.

8.3 ML Frameworks and Algorithms

We use standard Python programming interfaces such as scikit-learn, tensor-flow, and keras, for the implementation of the algorithms: logistic regression, random forest, multilayer perceptron, and convolutional neural networks.

9. Results

Responsible: Jose, Rafael, Thiago

9.1 Jet substructure analysis

The N-subjettiness τ_N [1] quantifies the capability of clustering the jet constituents in exactly N subjets. The ratio $\tau_{21} = \tau_2/\tau_1$ is a powerful discriminant between jets originating from hadronic V decays and from gluon and single-quark hadronization. Jets coming from hadronic W or Z decays are characterized by lower values of τ_{21}, given the two-prong substructure of the jet constituents. Fig. 6 shows the distribu-

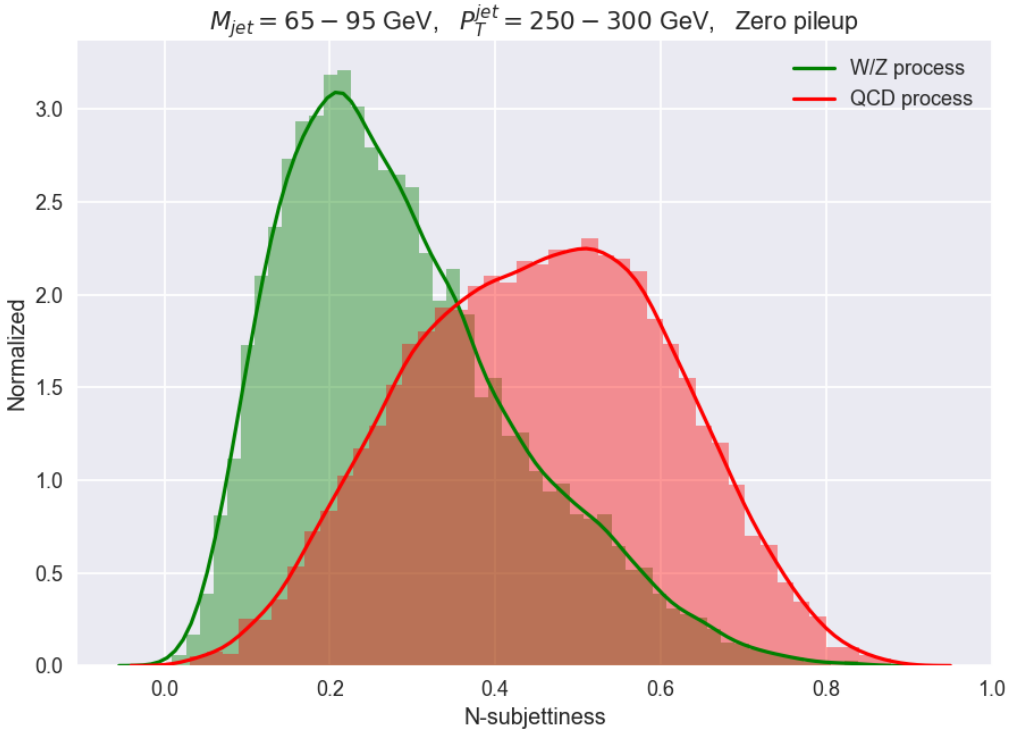

Figure 6: Distribution of the N-subjettiness τ_{21} variable for signal and background.

tion of the N-subjettiness τ_{21} for high energy jets coming from W/Z processes, and for similar jets coming from QCD processes.

Using logistic regression, we estimate a predicted probability of the form $g(z) = 1/(1+e^{-z})$ that assigns samples of outputs larger or equal to 0.5 to the positive class, and the rest to the negative class. The predicted probability returned by the logistic regression model is shown in Fig. 7.

We set $\tau_{21} = 0.37$ as the separation boundary between the positive and the

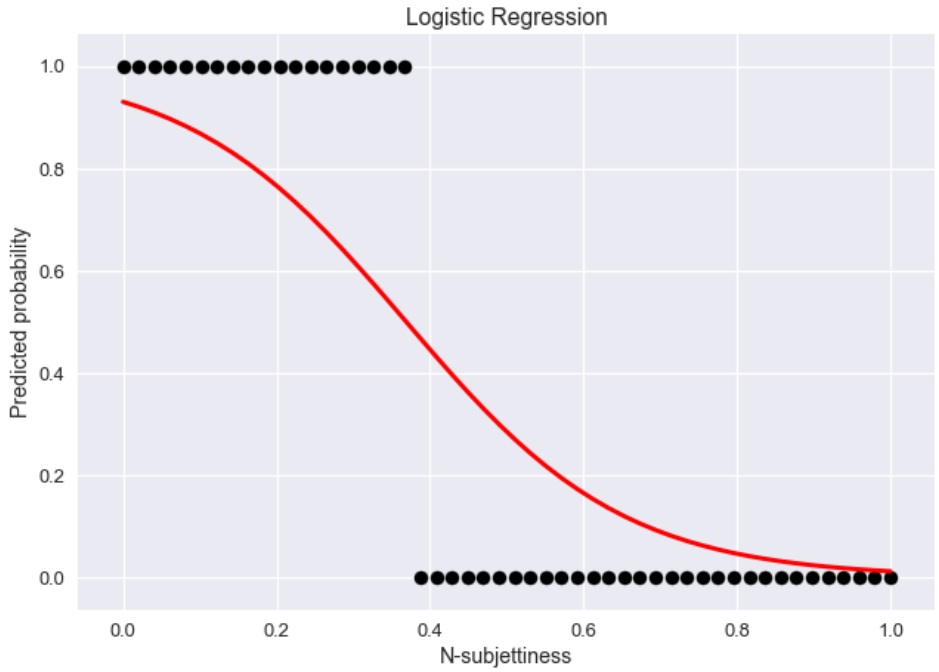

Figure 7: Predicted probability returned by the logistic regression model. The separation between the positive (signal) and the negative (background) class happens at the N-subjettiness value equal to $\tau_{21} = 0.37$.

negative class, as determined by the logistic regression. The true positive rate (TPR) and false positive rate (FPR) of the N-subjettiness classifier are:

$$TPR = 0.7105\,, \qquad FPR = 0.2717\,. \tag{3}$$

9.2 Machine Learning models

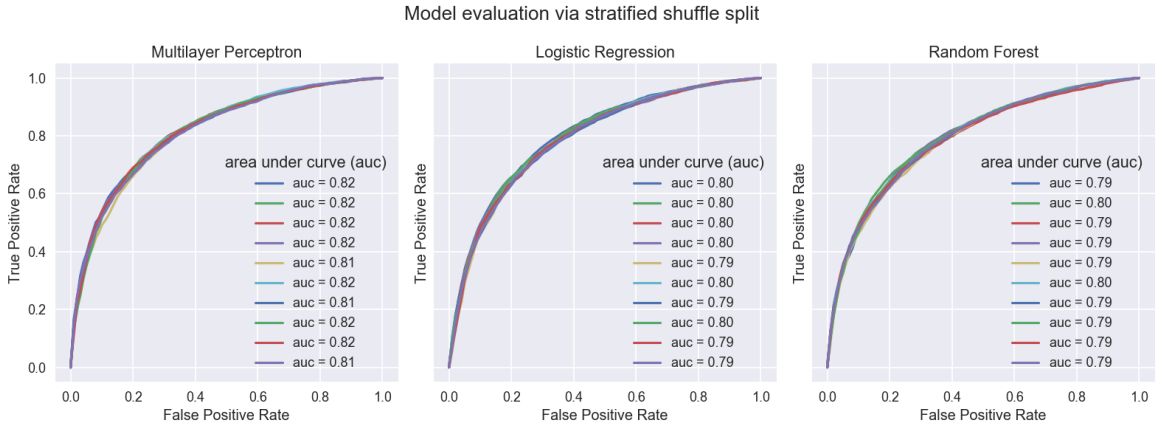

Figure 8: ROC curve obtained by different algorithms Multilayer Perceptron, Logistic Regression, and Random Forest.

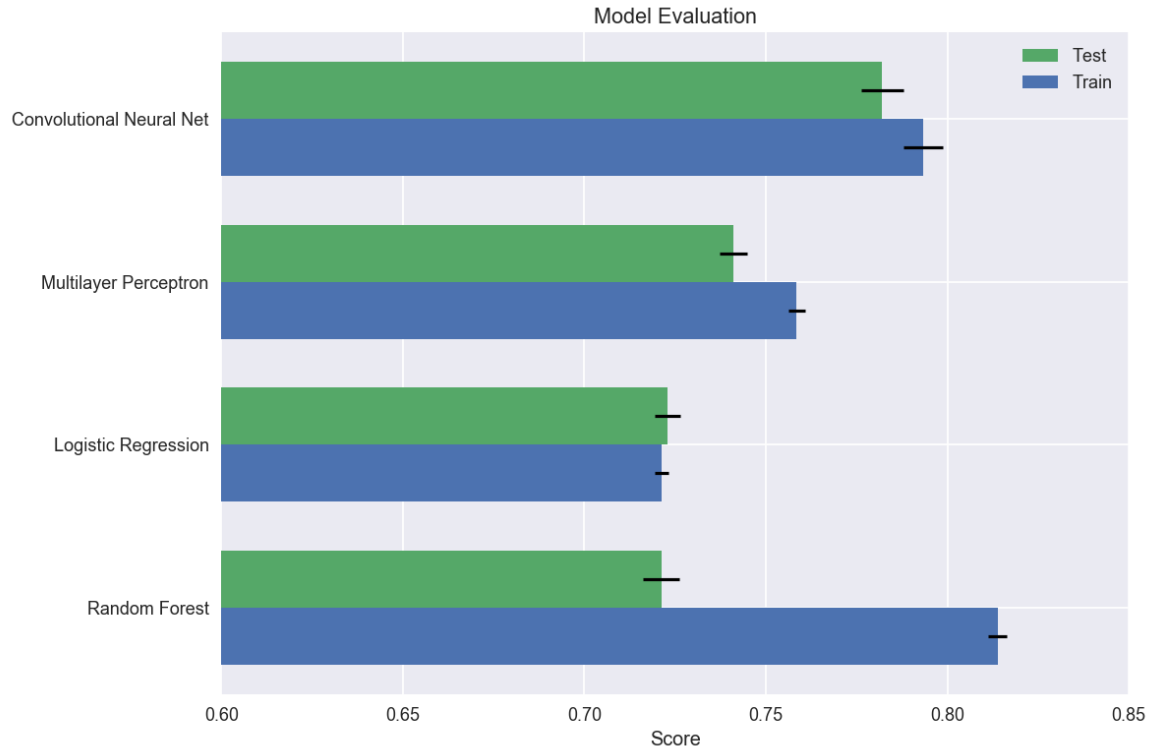

Figure 9: Accuracy score obtained by the different algorithms.

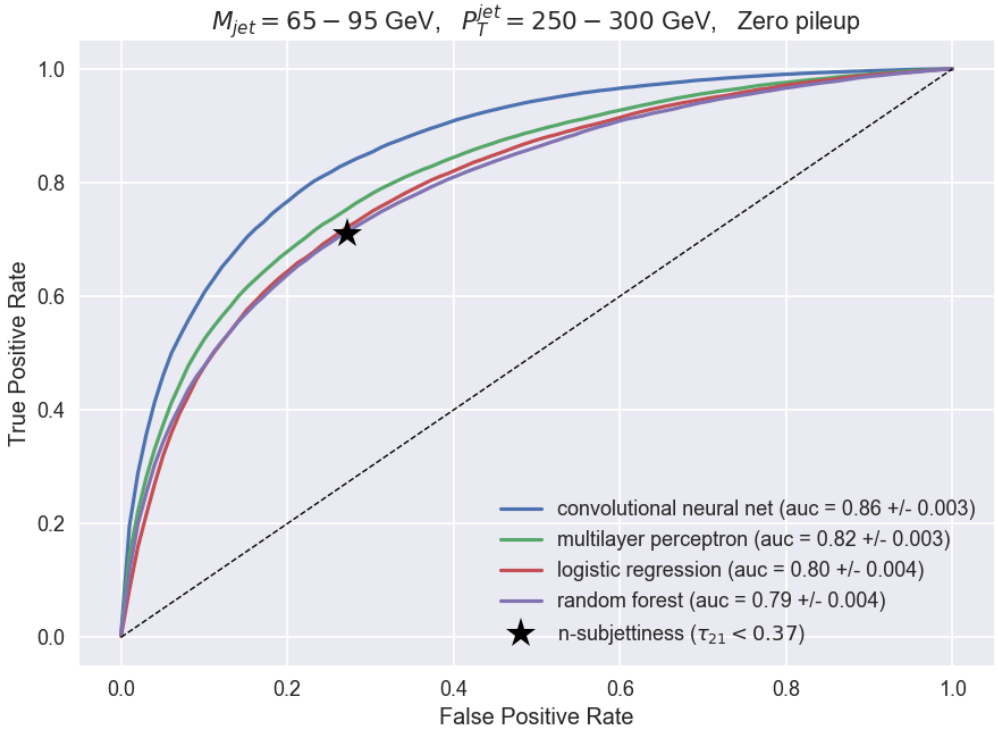

Figure 10: ROC curve obtained by the different algorithms.

10. Conclusion

Responsible: all

References

[1] Jesse Thaler and Ken Van Tilburg. Maximizing Boosted Top Identification by Minimizing N-subjettiness. *JHEP*, 02:093, 2012.