INTERNAL NOTE SIN–R XX/2017

Title of the Note: Use the Line Below if Continuation of the Title is Necessary

A. A. Author [1], B.B. Author [2]

[1] *University Uni*
[2] *University Duni*

Contents

1 Introduction 4
2 Machine Learning 5
2.1 What is Machine Learning 5
2.2 Machine Learning in High Energy Physics 5
2.3 Artificial Neural Networks 7
2.4 Random Forests 7
3 Pythia 8 8
3.1 Program Flow 8
3.2 A Pythia Hello World Program 8
3.3 The Output 10
4 Sample Generation 11
5 Jet Finding and Preprocessing 14
5.1 Jet Finding 14
5.2 Preprocessing 15
6 Machine Learning Frameworks 18
6.1 Python Neural Networks 18
6.2 R Logistic Regression 21
7 Results 23
7.1 Jet substructure analysis 23
7.2 Training and Evaluation of the ML Models 25
7.3 R Implementation 29
8 Conclusion 30

Figures

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. 15

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. 16

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 $\frac{1}{c}\sum_i E_{T,i}^2 = 1$. 17

4 RStudio screen. Obtained from [14]. 22

5 Distribution of the N-subjettiness τ_{21} variable for signal and background. 23

6 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 $\tau_{21} = 0.37$. 24

7 ROC curve obtained by different algorithms Multilayer Perceptron, Logistic Regression, and Random Forest. 25

8 ROC curve obtained with the convolutional neural network. 26

9 ROC curve obtained by different ML algorithms, compared with the N-subjettiness result. 27

10 Accuracy score obtained by different ML algorithms, using Python programming interfaces scikit-learn, tensorflow, and keras. The training (test) set corresponds to 70% (30%) of the sample. The uncertainty is estimated by taking the standard deviation over ten cross-validation splits. 28

11 Signal and background data means. Note that the jets are concentrated in the middle of the observed space, as expected. 29

Tables

1 Output for backgroung generation 13

2 Output for signal generation 13

3 Evaluation of Machine Learning Models 26

1. Introduction

2. Machine Learning

2.1 What is Machine Learning

Fundamentally Machine Learning is a mean of building a model of data [1]. It can be understood as the development of mathematical models which helps understanding und recoginizing patterns in data sets. Generally, these models have parameters which are fitted according to the observed data. Therefore, a data set works as the input for the model (basically a computer program) 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.

Basically, 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 [1]. An example of supervised learning use is classifying messages into categories spam or not, understood as labels. In this context, mapping algorithms can recognize words which are common in spam messages. A way to ”train” the model (ML algorithm) consists of using these spam words as inputs and associating them with the label spam through classification rules (set up of parameters). Therefore, when the model receive a brand-new dataset (unknown text, in this context) and it contains these previously learned words, certainly it will classify the text as a spam message according to the adjusted parameters.

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 [1].

2.2 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.

Jets are cone-like showers of hadrons originated from high energy quarks and

gluons or from decays which envolve Z^0, W^- and W^+ bosons. The Z^0 boson mediates decays which charge is not changed and W^- and W^+ mediate decays with changes of charge [2].

These decays generally produces highly boosted jets and these jets can merge into a single jet due to the large momentum and mass of the particles.

For example, in a collision between two protons p^+p^+, lots of particles can be generated through many kinds of processes. Even though, we can have specific ones which produce the referenced bosons.

For example, a quark down (d) with charge = $-1/3$ from one proton can turn on a quark up (u) with charge = $2/3$. If it occurs, the total charge must be the same. Hence, a particle whose charge is -1 should be produced. This way, we have a process mediated by a particle W^-, which carries a -1 charge and after the process is dissociated from the quark up produced. Furthermore, in the same collision, Z^0 and W^+ bosons can be generated through other processes. After that, these bosons can originate lots of gluons and also quarks up or quarks down again in a process called *hadronisation*, producing jets whose particles have large momentum and mass. Hier, this kind of jets will be called *Boson Jets*.

On the other hand, gluons and quarks are produced through other processes which do not envolve the referenced bosons still in the same collision. This kind of jets has high energies too. In this context, they will be called *QCD Jets*.

Talking about *High Energy Physics* context, in order to discover and study such systems, it is vital to discriminate boosted heavy particle jets which come from bosons and jets originated from quarks and gluons in QCD processes [3].

This way, determining which particle originates each jet can be understood as a machine learning classification problem.

One way to classify these jets is presented in [3]. In this paper, methods for preprocessing and discrimination are inspired by techniques in the field of computer vision and the task is similar to that of facial recognition. Thus, jet-specific analogs to the algorithms used in facial recognition were developed.

On the other hand, other studies have explored the possibility of reconstructing particles using image recognition techniques based on convolutional neural networks too [4]. This approach is described in the next sections.

The collision between two protos previously described can be simulated using the applications *Pythia8* and *Fastjet*. Thus, in the next section of there is an brief

introduction to Pythia8.

Responsible: Amanda

2.3 Artificial Neural Networks

The area of Neural Network tries to mime the behaviour of the human brain, that is a highly complex parallel system, consisting of about 10^{11} neurons of several types and shapes [5]. From its first steps, the area of Artificial Neural Networks has been inspired by such complexity of the brain by acknowledging that its ability to make computations and recognize patterns can outperform even the most recent super-computers, e.g., the human brain is able to identify a familiar face on an unfamiliar scene in under 200ms, while performing less complicated tasks takes a lot longer on modern computers [6].

It is largely believed that the computational power delivered by such models comes from its high degree of parallelism and the capacity to *learn*, making it possible for these system to generalize and provide good approximate answers to complex problems [6].

The studies on the field of Neural Network started with studies of a single Neuron - the basic brain structure. In 1940, the first machinary model of neuron was proposed by McCulloch and Pitts [7]. They proposed to model the Neuron as a simple activation machine, i.e, a threshold device that was able to perform simple logical functions - *conjunctions* and *disjunctions*. Still, the procedure for *learning* was yet to be defined.

It was only in 1949, that Hebb proposed a theory on how the learning over time affects the interaction between neurons [8]. In 1952, the model that approaches the one currently being used was developed by Hodgin and Huxley in [9]. They included the ideas of neuronal firing and the effects of the threshold on the neural signal propagation.

2.3.1 Perceptron

2.3.2 Back-propagation training procedure

2.4 Random Forests

3. Pythia 8

Pythia 8 is a tool for generation of high-energy collisions. The particles are produced in vacuum.It 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. Pythia also has a set of utilities and interfaces to external programs. Thus, it is possible to run it together with other 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 [10].

3.1 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 swittched 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

3.2 A Pythia 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 [11].

- **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.

3.3 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 [10]

Responsible: Amanda

4. Sample Generation

As it was described in Subsection 2.2, a collision between two protons where *Boson Jets* and *QCD Jets* are mixed up can be simulated through *Pythia8*. In this section, there is an example of a *Pythia8* code which simulates this situation:

```
int main(int argc, char* argv[])
{
    // Basic run parameters
    int nEvent    = 1000;
    int nEventPU  = 0;
    int nListJets = 3;
    int processType =  0; //0 = background, 1 = signal

    // LHC parameters
    double sqrtsInGeV = 13000.0;  //LHC collision energy
    double meanPU = 0.0;
    double _minJetMass = 65;
    double _maxJetMass = 95;
    double _minJetPt = 200; //Range of transverse momentum
    double _maxJetPt = 250;
    ...
    Event &event = pythia.event;
    Event &process = pythia.process;
    Event &eventPU = pythiaPU.event;
    ...
    // Process selection.
    if(processType == 0) {
        pythia.readString("HardQCD:all = on"); //QCD Jets
        }
    if(processType == 1) {
        pythia.readString("WeakDoubleBoson:ffbar2ZW = on"); //Boson Jets
        pythia.readString("23:onMode = off");
        pythia.readString("23:onIfAny = 12 14 16");
        pythia.readString("24:onMode = off");
        pythia.readString("24:onIfAny = 1 2 3 4 5");
        }
      ...
```

```
      pythiaPU.readString("SoftQCD:nonDiffractive = on"); //Pile Up
      ...
    pythia.init();
    pythiaPU.init();
      ...
    // Event loop and Pile Up Event generation loop
    for (int iEvent = 0; iEvent < nEvent; ++iEvent)
    {
      ...
    for (int iEventPU = 0; iEventPU != nEventPU; ++iEventPU)
    {
      ...
    }
  }
     // Statistics. Histograms.
     pythia.stat();

    // Done.
    return 0;
}
```

Hier we have the simulation of 1000 events, including the generation of *Boson Jets*, *QCD Jets* and Pile Up events. Pile Up events are the production of particles whose energy is not so high during the collision. Thus, it can be understood as another kind of background. The choice of which jets will be generated depends basically on the parameter*ProcessType*.

Hence, when the $ProcessType = 0$, only QCD Jets will be generated. As we are interested mainly in Z^0 and W^+ bosons identification, naturally we call QCD Jets *background*. In this case, only gluon particles are produced and the program output is shown in Table 1.

On the other hand, if the $ProcessType = 1$, Z^0 and W^+ bosons, quarks and another particles are produced. Hence, this kind of process is called *signal*. This output is shown in Table 2.

Table 1: Output for backgroung generation

id	name	status	mothers	daughters	colors	px	py	pz	e
90	(system)	-11	0 0	0 0	0 0	0.000	0.000	0.000	13000.00
2212	(p^+)	-12	0 0	3 0	0 0	0.000	0.000	6500.000	6500.000
2212	(p^+)	-12	0 0	4 0	0 0	0.000	0.000	-6500.000	6500.000
21	(g)	-21	1 0	5 6	101 102	0.000	0.000	69.018	69.018
21	(g)	-21	2 0	5 6	103 104	0.000	0.000	-361.302	-361.302
21	g	23	3 4	0 0	101 104	-66.853	-143.051	-148.714	-148.714
21	g	23	3 4	0 0	103 102	66.853	143.051	-143.570	-143.570

Table 2: Output for signal generation

id	name	status	mothers	daughters	colors	px	py	pz	e
90	(system)	-11	0 0	0 0	0 0	0.000	0.000	0.000	13000.00
2212	(p^+)	-12	0 0	3 0	0 0	0.000	0.000	6500.000	6500.000
2212	(p^+)	-12	0 0	4 0	0 0	0.000	0.000	-6500.000	6500.000
21	(g)	-21	1 0	5 6	101 102	0.000	0.000	69.018	69.018
21	(g)	-21	2 0	5 6	103 104	0.000	0.000	-361.302	-361.302
21	g	23	3 4	0 0	101 104	-66.853	-143.051	-148.714	-148.714
21	g	23	3 4	0 0	103 102	66.853	143.051	-143.570	-143.570

Responsible: Thiago, Amanda

5. 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.

5.1 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.

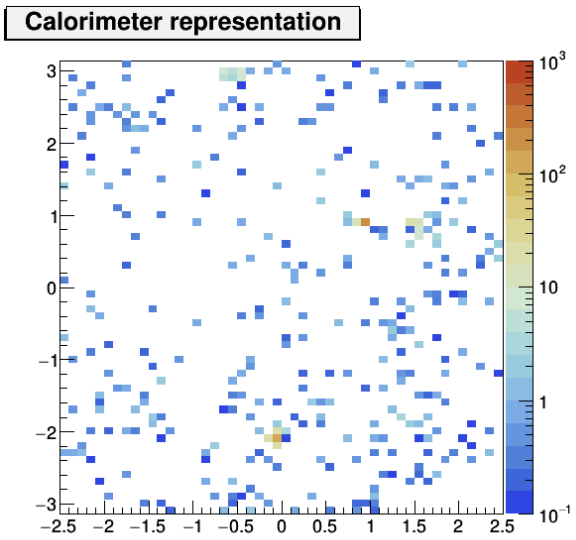


Figure 1: Example of calorimeter representation of event. The x-axis of the plot represents the η coordinate, while the y-axis represents the ϕ coordinate. The z-axis represents the total E_T in the each calorimeter cell. The event is a dijet (background) event, where the leading jet has p_T = 258 GeV,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

5.2 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

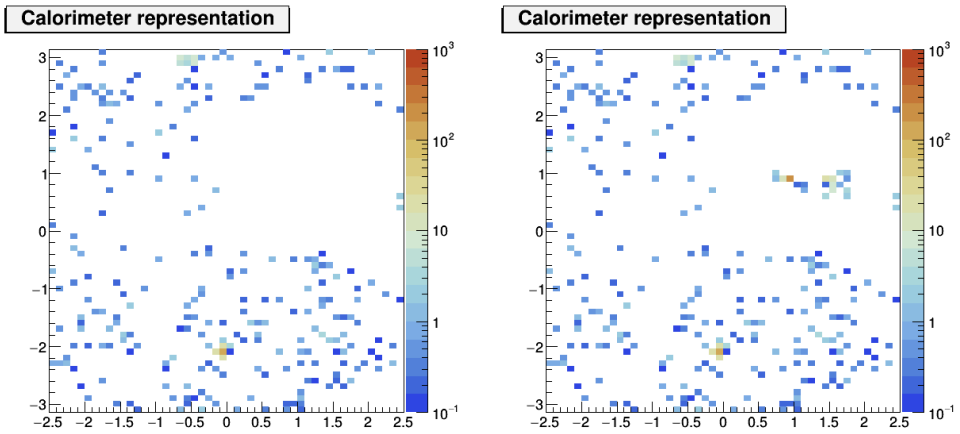


Figure 2: Example of calorimeter representation of event. The x-axis of the plot represents the η coordinate, while the y-axis represents the ϕ coordinate. The z-axis represents the total E_T in the each calorimeter cell. In the left plot, the cells located around the leading jet are zeroed out, while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

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

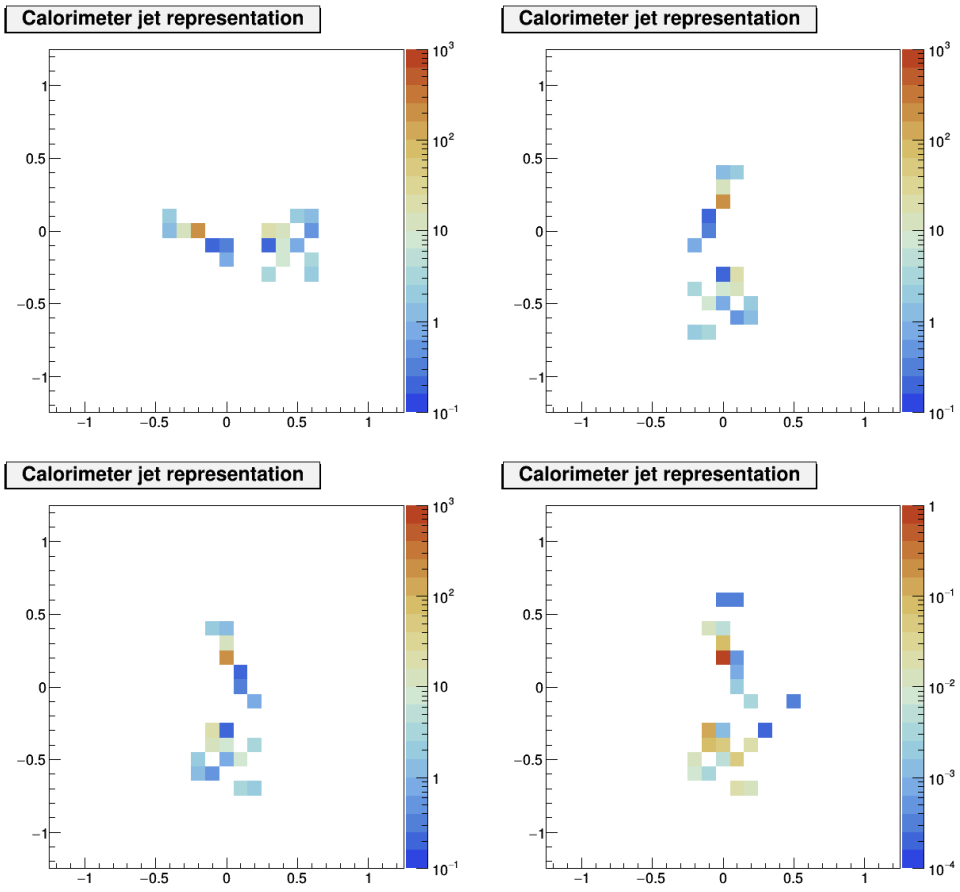


Figure 3: Example of calorimeter representation of jet. The x-axis of the plot represents the η coordinate, while the y-axis represents the ϕ coordinate. The z-axis represents the total E_T in the each calorimeter cell. In the top left plot, the coordinates have been changed such that the centroid cell 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 $\frac{1}{c}\sum_i E_{T,i}^2 = 1$.

6. Machine Learning Frameworks

Responsible: Jose, Rafael

6.1 Python Neural Networks

In this section we aim to describe a procedure to deploy a basic python software stack to develop Artificial Neural Networks. We will first describe the procedure used to install the needed software, then we will describe briefly how a simple application can be built on top of a high level framework.

6.1.1 Environment Setup

Managing dependencies on a data analysis project can be a hard task. Building a stable environment depends on the ability to install the correct version of the software tools.

For this purpose, tools like *Anaconda* make possible for users to create *Virtual Python Environments*. These environments provide an isolation layer between two python setups, i.e., it allows several python installations to to be used in the same operating system.

The basic idea that allow the isolation is that the python virtual machine looks for its libraries at specific paths while executing a program. Tools like Anaconda and *VirtualEnv* override specific environment variables, indicating the correct python installation path and so on.

The Anaconda installation is very straight forward and one only needs to download the installation software[1] and run it:

```
# bash Anaconda.sh
```

The software can be installed in user space, therefore no administrative rights are needed to install it. After the installation the `conda` command line there should have been made available to the user.

The `conda` command line allows to switch between environments, create environments and install different versions of libraries and modules. In order to gather information about the Anaconda installation one has to perform the following command:

[1] https://www.continuum.io/downloads

```
# conda info
  Current Anaconda install:
              platform : osx-64
  conda command version : 1.3.2
         root directory : /Users/rocknroll/anaconda
         default prefix : /Users/rocknroll/anaconda
           channel URLS : ['http://repo.continuum.io/pkgs/free/osx-64/']
  environment locations : ['/Users/rocknroll/anaconda/envs']
```

For creating a new environment, one must perform the `conda create` command and specify the name for the environment and the list of the basic software that should be installed at the environment. For instance, the following command will create a virtual environment named *python35* with python version 3.5 installed in it.

```
# conda create -n python35 python=3.5
Solving package specifications: .
Package plan for installation in environment /Users/rocknroll/envs/
python35:
The following NEW packages will be INSTALLED:
    openssl:    1.0.2l-0
    pip:        9.0.1-py35_1
    python:     3.5.3-1
    readline:   6.2-2
    setuptools: 27.2.0-py35_0
    sqlite:     3.13.0-0
    tk:         8.5.18-0
    wheel:      0.29.0-py35_0
    xz:         5.2.2-1
    zlib:       1.2.8-3

Proceed ([y]/n)?
```

After the creation of the environment, one should issue the *activate* command:

```
# source activate python35
# which python
~/envs/python35/bin/python
```

6.1.2 The Keras Framework

Keras is a high-level neural networks API, written in Python (versions 2.7 through 3.6) and capable of running on top of either TensorFlow or Theano.

The initial building block of Keras is a model, and the simplest model is called sequential. A sequential Keras model is a linear pipeline (a stack) of **neural networks layers**. An example of a Neural Net in Keras: The code below defines a single (dense) layer with 12 artificial neurons, and 8 input variables (**features**):

```
from keras.models import Sequential
model = Sequential()
model.add(Dense(12, input_dim=8, kernel_initializer='random_uniform'))
```

Each neuron can be initialized with specific weights. Keras provides a few choices, the most common of which are [2]:

- `'random_uniform'`: Weights are initialized to uniformly random small values in `(-0.05, 0.05)`
- `'random_normal'`: Weights are initialized according to a Gaussian, with a zero mean and small standard deviation of `0.05`.
- `zero`: All weights are initialized to zero.

Before training a model it is necessary to configure the learning process. This is done by using the `compile` method. This method receives three arguments:

1. An Optimizer such as *rmsprop* or *adam*;
2. Objective Function - "loss function". This is the objective that the model will try to minimize;
3. A list of metrics: `metrics=['accuracy']`.

For a multi-class classification problem

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy'])
```

For a binary classification problem

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy',
              metrics=['accuracy'])
```

[2] https://keras.io/initializations/

For regression problem:

```
model.compile(optimizer='rmsprop', loss='mse')
```

For training a model use the `fit` function. For binary classification:

```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels, epochs=10, batch_size=32)
```

For categorical classification:

```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
              metrics=['accuracy']))
model.fit(data, labels, epochs=10, batch_size=32)
```

6.1.3 Usage Example - Recognizing Hand-Written Digits

6.2 R Logistic Regression

6.2.1 Environment Setup

Setting up an environment in R has some advantages. Due to the strong relation of the language to statistical applications, many of the resources needed for machine learning such as training, testing and evaluation of the algorithm performance are available out-of-the-box. In GNU/Linux systems, the language generally is available through the package managers. In Debian like systems, for example, it can be installed using the following command:

```
# apt-get install r-base r-base-dev
```

Besides the language, packages which extend the functionalities could be installed using the CRAN (Comprehensive R Archive Network) [12]. This can be done from the R console, opening it (typing *R* in your terminal) and using the install package command.

```
R> install.package("package_name")
```

Another very popular resource for developing in R and managing the packages is RStudio [13], a free and open-source integrated development environment (IDE) for R. This tool provides an multi-platform (GNU/Linux, Mac and Windows) environment for developing projects, presenting a code editor, a R console, an workspace track and a plot windows, as shows on Figure 4.

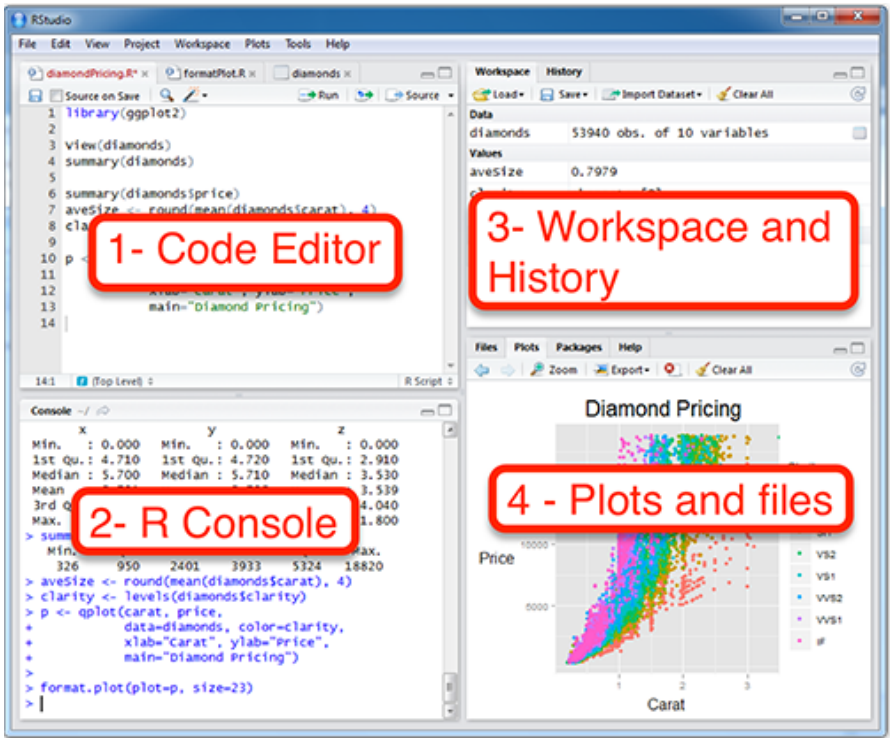


Figure 4: RStudio screen. Obtained from [14].

6.2.2 Usage Example

7. Results

Responsible: Jose, Rafael, Thiago, Vitor

7.1 Jet substructure analysis

The N-subjettiness τ_N [15] 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. 5 shows the distribu-

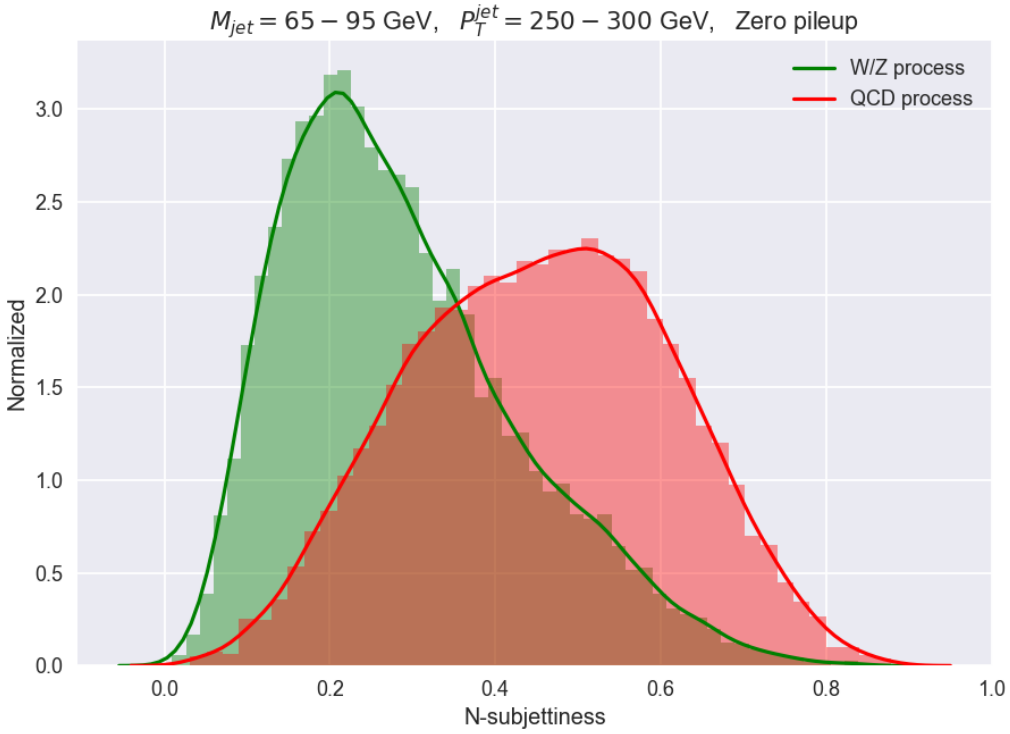


Figure 5: Distribution of the N-subjettiness τ_{21} variable for signal and background.

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. 6.

As a result of the logistic regression, a separation boundary at $\tau_{21} = 0.37$ is set

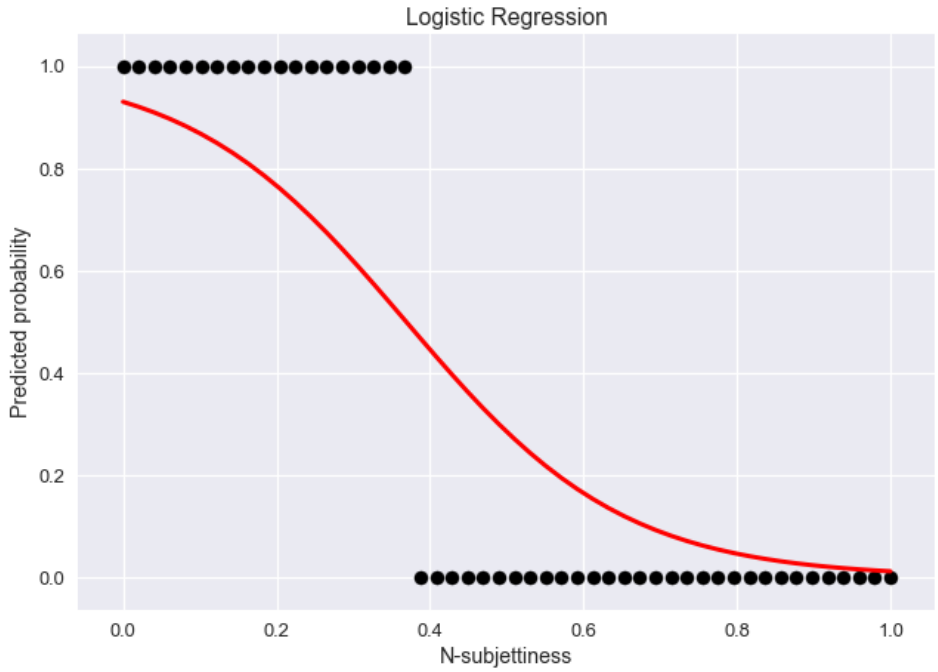


Figure 6: Predicted probability returned by the logistic regression model. The separation between the positive (signal) and the negative (background) class happens at the N-subjettiness value equal to $\tau_{21} = 0.37$.

between the positive and the negative class. The true positive rate (TPR) and false positive rate (FPR) of the N-subjettiness classifier, obtained by counting the number of events at the left and right of the boundary $\tau_{21} = 0.37$, are given by:

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

7.2 Training and Evaluation of the ML Models

A good way to evaluate a model is to use cross-validation. Starting from two separate data samples, one for signal and other for background, the workflow requires the concatenation of the two samples into a single data frame. Consequently, an stratified-shuffle-split procedure is applied to split the data frame into training (70%) and testing (30%) sets. In order to avoid random effects and unbalanced distribution of the classes in the split, the procedure is repeated 10 times.

The receiver operating characteristic (ROC) curve is used to evaluate the performance of a model. It consists of a plot of the true positive rate in the y-axis versus the false positive rate in the x-axis. For each cross-validation split, a ROC curve was drawn as presented in Fig. 7. Among the algorithms Multilayer Perceptron, Logistic Regression, and Random Forest, the best model displaying the higher area under curve (auc) is the Multilayer Perceptron.

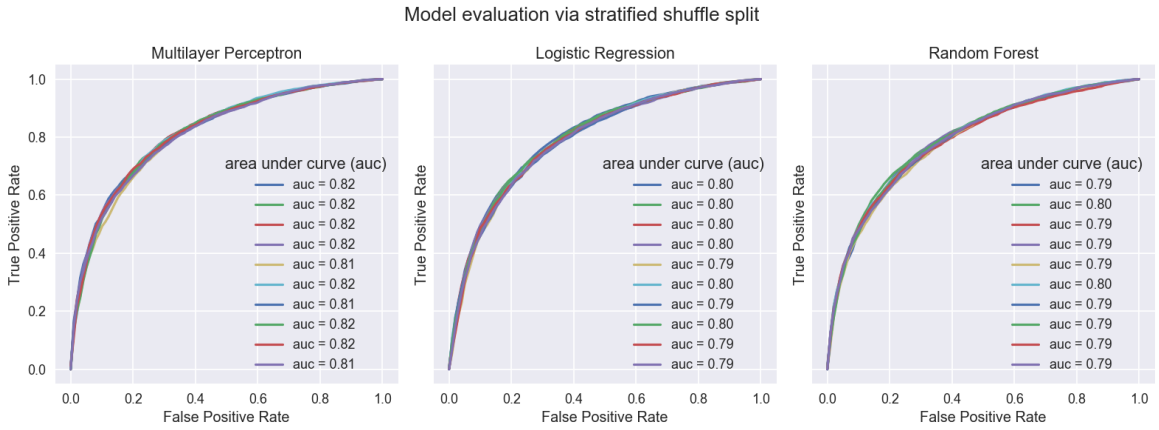


Figure 7: ROC curve obtained by different algorithms Multilayer Perceptron, Logistic Regression, and Random Forest.

The same cross-validation procedure was applied for the most complex model, that is the convolutional neural network. The ROC curves for the different splits are shown in Fig. 8.

Finally, all ROC curves are summarized in Fig. 9, as well as in Table 3.

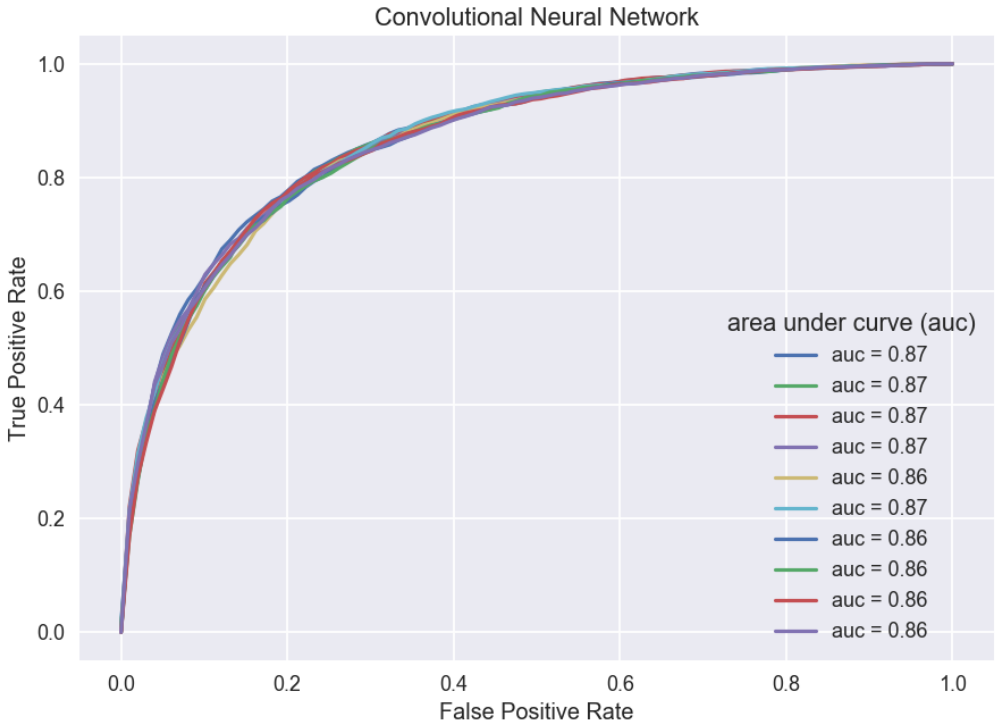


Figure 8: ROC curve obtained with the convolutional neural network.

Table 3: Evaluation of Machine Learning Models

Model	**AUC**	**Description**
Convolutional Neural Net	0.86 ± 0.003	3 conv. layers, dropout of 25%
Multilayer Perceptron	0.82 ± 0.003	1 hidden layer, 5 neurons
Logistic Regression	0.80 ± 0.004	liblinear solver, L2 regularization
Random Forest	0.79 ± 0.004	Number of trees 24, depth 9

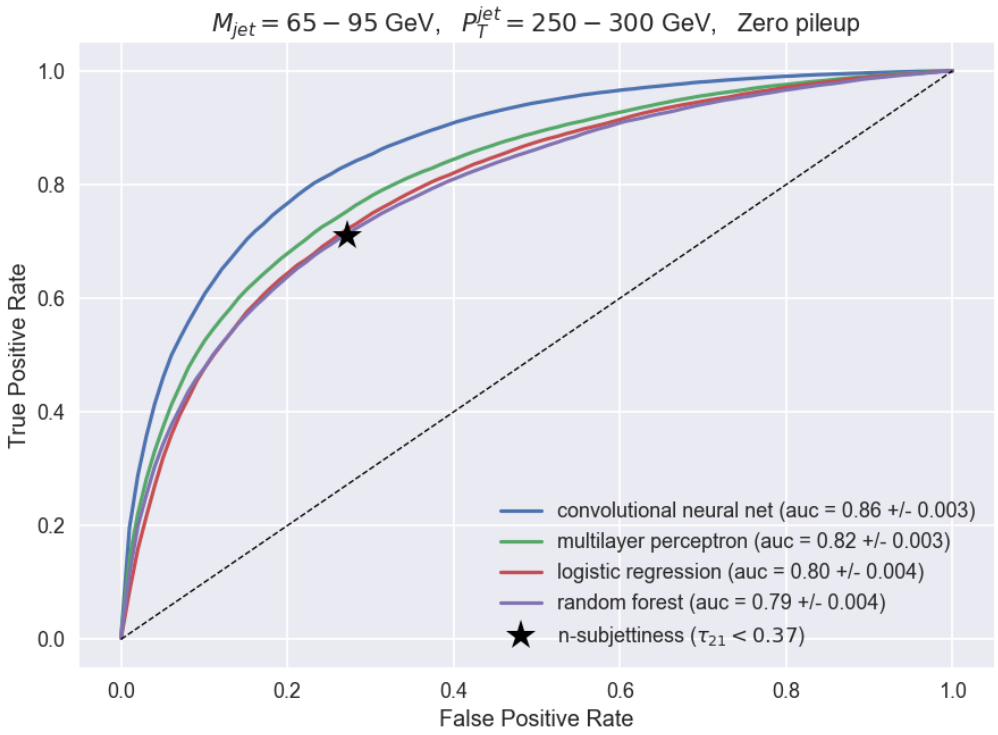


Figure 9: ROC curve obtained by different ML algorithms, compared with the N-subjettiness result.

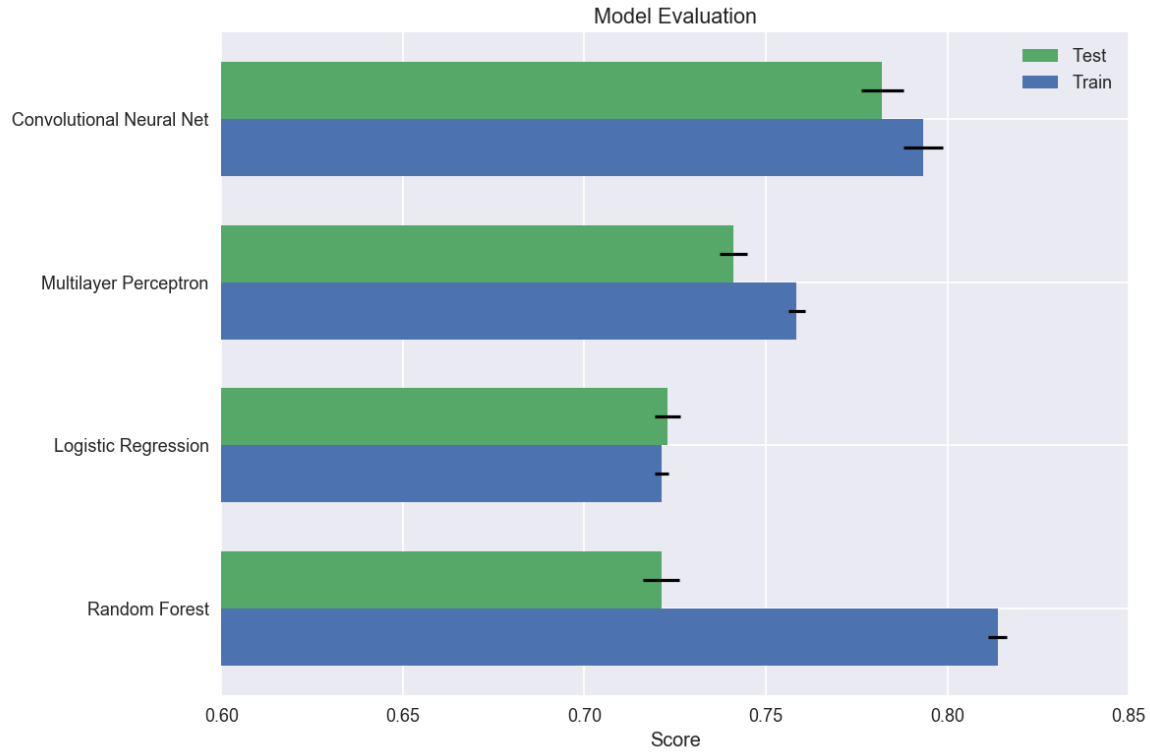
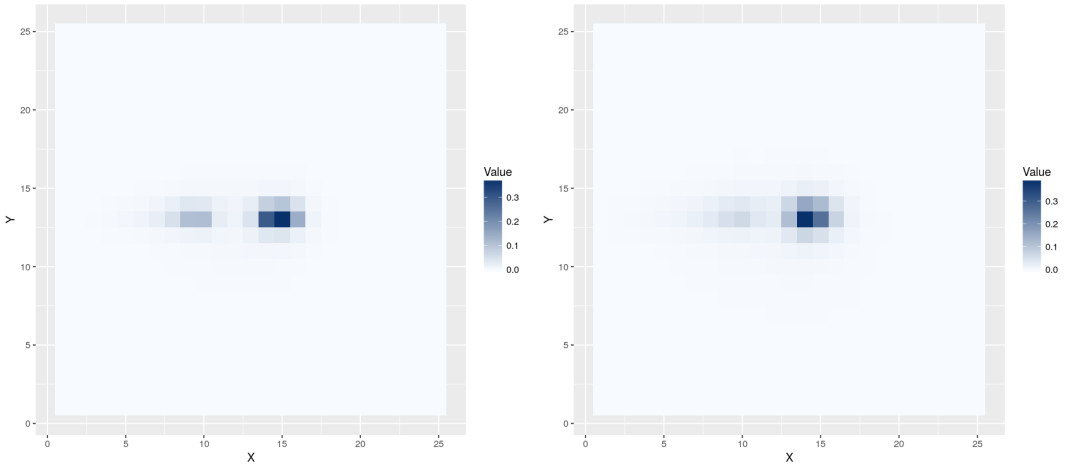


Figure 10: Accuracy score obtained by different ML algorithms, using Python programming interfaces scikit-learn, tensorflow, and keras. The training (test) set corresponds to 70% (30%) of the sample. The uncertainty is estimated by taking the standard deviation over ten cross-validation splits.

7.3 R Implementation

Another very popular tool for dealing with machine learning issues is R, which is a language and environment for statistical computing and graphics [16]. In order to have a glimpse of its syntax and particularities, an implementation of the logistic regression algorithm was done in R trying to replicate as much as possible the steps done in Python. The results can be checked in details at its github repository [17].

Firstly, both signal and background data were imported using the proper commands. The mean of every pixel was taken so it would be possible to check the average behavior of signal and background data. The result can be seen on Figure 11.



(a) Signal Mean (b) Background Mean

Figure 11: Signal and background data means. Note that the jets are concentrated in the middle of the observed space, as expected.

Since the data set was the same as the previously used, the algorithm is expected to present a similar performance of the logistic regression done in python. After running the trained algorithm in the testing set, an accuracy of 72.6% was observed.

This next paragraph should be checked and referenced with the proper reference.

Although R is a popular choice for machine learning issues, limitations in multicore processing could be a potential disadvantage when comparing with python.

8. Conclusion

Responsible: all

References

[1] STHDA. Running rstudio and setting up your working directory - easy r programming. http://www.sthda.com/english/wiki/running-rstudio-and-setting-up-your-working-directory-easy-r-programm

[2] Jake VanderPlas. Python data science handbook, 2016.

[3] Chris Quigg. Elementary particles and forces. *Scientific American*, 252(4):84–94, 1985.

[4] Josh Cogan, Michael Kagan, Emanuel Strauss, and Ariel Schwartzman. Jet-images: computer vision inspired techniques for jet tagging. *Journal of High Energy Physics*, 2015(2):118, 2015.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[6] Ke-Lin Du and Madisetti NS Swamy. *Neural networks and statistical learning*. Springer Science & Business Media, 2013.

[7] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Upper Saddle River, NJ, USA:, 2009.

[8] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[9] Donald Olding Hebb. *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949.

[10] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[11] T. Sjostrand et al. A Brief Introduction to PYTHIA 8.1. *http://home.thep.lu.se/ torbjorn/pythia81html/Welcome.html*, 2007.

[12] T. Sjostrand et al. PYTHIA 8 Worksheet. *http://home.thep.lu.se/ torbjorn/pythia81html/Welcome.html*, 2014.

[13] The comprehensive r archive network. https://cran.r-project.org/.

[14] R studio. https://www.rstudio.com/.

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

[16] The R Foundation. What is R? https://www.r-project.org/about.html.

[17] Sao Paulo Research and Analysis Center. Jet Classification in R. https://github.com/SPRACE/jet-classification-R.