ASTROCENT

# Machine Learning approach for Pion Identification

Sarthak Choudhary
sarthak@camk.edu.pl
(Doctoral Student)
AstroCeNT

June 16, 2021

NICOLAUS COPERNICUS
ASTRONOMICAL CENTER
OF THE POLISH ACADEMY OF SCIENCES

A report on project undertaken in partial fulfilment of the course on **Statistics Under Artificial Intelligence**

Submitted to

Prof. Jordi Batalla

# Contents

# Chapter 1

# Introduction

Particle physics experiments create enormous amount of data. The Large Hadron Collider(LHC) generated 15 petabytes of data for each year of its run. In the upcoming years several particle physics experiments such as Darkside-20k and DUNE will come online, generating even more data. Analysis of such huge data sets presents novel challenges as the classical methods of data analysis are quite laborious and expensive in terms of human hours. Machine Learning techniques are being considered to aid in the data analysis. In this report we present a Machine Learning approach for identification of pions from large volumes of data generated by particle interaction experiments.

## 1.1 Pions

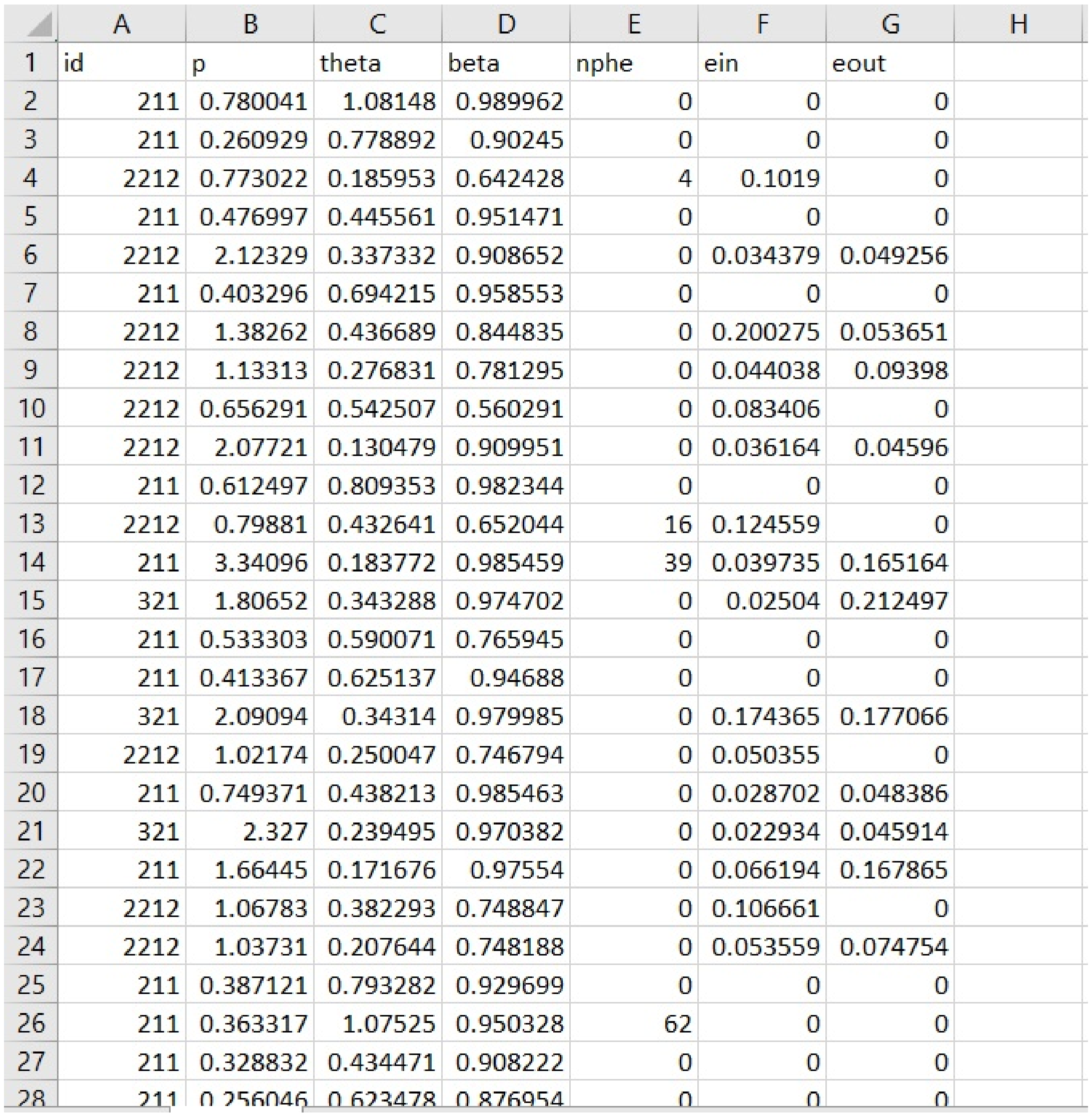
**What are pions and why do we need to identify them?**

A pi meson or Pion is a subatomic particle consisting of a quark and an antiquark. They are unstable and have very short life spans.
In order to study properties of sub-nuclear particle, it is important to correctly identify the particles which are generated in particle accelerators. Pions are the most abundant particle species in a proton-proton collision such as those which occur in LHC. Pions can be either charged ($\pi^+$ or $\pi^-$) or neutral ($\pi^0$). Neutral pions promptly decay into photon pairs subsequently generating compact showers which are easily detected by Calorimeters in the detector. However, charged pions generate irregular showers which get absorbed only after passing through several layers of Calorimeter detector. For accurate energy calibration it is necessary to identify which particles are depositing how much energy and in which specific parts of the detector. Since, pions are the most abundant of daughter particles in a proton-proton collision, pion identification is critical for energy calibration of the detector.

## 1.2 Data

Since the scope of this project was to better understand Machine Learning techniques available for such a task, instead of using data from particle interactions in a particle accelerator we used simulation data available on kaggle. The interaction were generated by Geant4, a standard tool used to simulate energy particle interactions.
The data set can be downloaded from this link: `https://www.kaggle.com/naharrison/particle-identification-from-detector-responses?select=pid-5M.csv`.

The datasheet in figure 1.1 shows snippet of simulated data. Rows refer to individual hits and the columns shows particle properties, e.g., ID (particle type), p (momentum in GeV/c), nphe (number of photo electrons), theta in Radians, beta, ein (inner energy in GeV) and eout (outer energy in GeV) respectively.
Values in the ID column can be used to identify different particle types using the following guide: positron (-11), pion (211), kaon (321), and proton (2212).
The full data set has 5 million rows.
In the subsequent chapter, we present a Machine Learning based approach to identify Pions. The simulation data has been used to train a Machine Learning model from which we can classify Pions from other particle types.



| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | id | p | theta | beta | nphe | ein | eout | |
| 2 | 211 | 0.780041 | 1.08148 | 0.989962 | 0 | 0 | 0 | |
| 3 | 211 | 0.260929 | 0.778892 | 0.90245 | 0 | 0 | 0 | |
| 4 | 2212 | 0.773022 | 0.185953 | 0.642428 | 4 | 0.1019 | 0 | |
| 5 | 211 | 0.476997 | 0.445561 | 0.951471 | 0 | 0 | 0 | |
| 6 | 2212 | 2.12329 | 0.337332 | 0.908652 | 0 | 0.034379 | 0.049256 | |
| 7 | 211 | 0.403296 | 0.694215 | 0.958553 | 0 | 0 | 0 | |
| 8 | 2212 | 1.38262 | 0.436689 | 0.844835 | 0 | 0.200275 | 0.053651 | |
| 9 | 2212 | 1.13313 | 0.276831 | 0.781295 | 0 | 0.044038 | 0.09398 | |
| 10 | 2212 | 0.656291 | 0.542507 | 0.560291 | 0 | 0.083406 | 0 | |
| 11 | 2212 | 2.07721 | 0.130479 | 0.909951 | 0 | 0.036164 | 0.04596 | |
| 12 | 211 | 0.612497 | 0.809353 | 0.982344 | 0 | 0 | 0 | |
| 13 | 2212 | 0.79881 | 0.432641 | 0.652044 | 16 | 0.124559 | 0 | |
| 14 | 211 | 3.34096 | 0.183772 | 0.985459 | 39 | 0.039735 | 0.165164 | |
| 15 | 321 | 1.80652 | 0.343288 | 0.974702 | 0 | 0.02504 | 0.212497 | |
| 16 | 211 | 0.533303 | 0.590071 | 0.765945 | 0 | 0 | 0 | |
| 17 | 211 | 0.413367 | 0.625137 | 0.94688 | 0 | 0 | 0 | |
| 18 | 321 | 2.09094 | 0.34314 | 0.979985 | 0 | 0.174365 | 0.177066 | |
| 19 | 2212 | 1.02174 | 0.250047 | 0.746794 | 0 | 0.050355 | 0 | |
| 20 | 211 | 0.749371 | 0.438213 | 0.985463 | 0 | 0.028702 | 0.048386 | |
| 21 | 321 | 2.327 | 0.239495 | 0.970382 | 0 | 0.022934 | 0.045914 | |
| 22 | 211 | 1.66445 | 0.171676 | 0.97554 | 0 | 0.066194 | 0.167865 | |
| 23 | 2212 | 1.06783 | 0.382293 | 0.748847 | 0 | 0.106661 | 0 | |
| 24 | 2212 | 1.03731 | 0.207644 | 0.748188 | 0 | 0.053559 | 0.074754 | |
| 25 | 211 | 0.387121 | 0.793282 | 0.929699 | 0 | 0 | 0 | |
| 26 | 211 | 0.363317 | 1.07525 | 0.950328 | 62 | 0 | 0 | |
| 27 | 211 | 0.328832 | 0.434471 | 0.908222 | 0 | 0 | 0 | |
| 28 | 211 | 0.256046 | 0.623478 | 0.876954 | 0 | 0 | 0 | |

Figure 1.1: Particle data

# Chapter 2

# Machine Learning

### What is Machine Learning?

Machine learning (ML) is the study of computer algorithms that improve automatically through experience and by the use of data.
The applications of Machine Learning include but are not limited to regression, classification, and clustering of data.
Three types of Machine Learning algorithm are :

1. Supervised Learning
2. Unsupervised Learning
3. Reinforced Learning

The difference between Supervised and Unsupervised learning is that in Supervised Learning output labels are known to the 'machine' while in Unsupervised Learning output labels are not known to the machine.
Reinforced Learning is another approach where output labels are not known to the machine and it is rewarded (penalized) for predicting the right (wrong) label. Since we want to label events according to a priori knowledge about particle characteristics, e.g., mass, momentum and velocities, we will use a supervised machine learning approach.
First we need to build a machine learning model then we will train the model with our selected data set. Finally we will test the accuracy of our model.

## 2.1 Building a Machine Learning model

The problem of identifying whether a particle is a Pion or not can also be stated as a binary classification problem. To successfully classify whether a particle is a pion or not we need a reliable model. In order to build a machine learning model we need two components, namely, data and an algorithm. Data set has already been discussed in chapter 1.

### Algorithm

The Data has several parameters which could possibly be used for classification but not all features would necessarily be important for classification. A machine learning algorithm known as Decision Trees helps to select important features for this classification.
As an advance approach we use an algorithm known as Random Forest which consist of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest gives a class prediction. Thus the class with the most votes becomes our model's prediction. The algorithm has been visually depicted in fig 2.1.
The philosophy behind Random Forest is that a large number of relatively uncorrelated models operating as a committee will outperform any of the individual constituent models.

## 2.2 Python Program

The python program was developed in jupyter lab. The notebook with python code is displayed below:

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from  sklearn.model_selection import train_test_split
import seaborn as sns
```

In [2]:
```
file_location = r'D:\D Downloads\Coursework Projects\Machine Learning Project\pid-5M
```

In [3]:
```
file_data = pd.read_csv(file_location)
```

In [4]:
```
file_data.head(100)
```

Out[4]:

| | id | p | theta | beta | nphe | ein | eout |
|---|---|---|---|---|---|---|---|
| **0** | 211 | 0.780041 | 1.081480 | 0.989962 | 0 | 0.000000 | 0.000000 |
| **1** | 211 | 0.260929 | 0.778892 | 0.902450 | 0 | 0.000000 | 0.000000 |
| **2** | 2212 | 0.773022 | 0.185953 | 0.642428 | 4 | 0.101900 | 0.000000 |
| **3** | 211 | 0.476997 | 0.445561 | 0.951471 | 0 | 0.000000 | 0.000000 |
| **4** | 2212 | 2.123290 | 0.337332 | 0.908652 | 0 | 0.034379 | 0.049256 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **95** | 2212 | 1.360950 | 0.147539 | 0.824000 | 0 | 0.000000 | 0.000000 |
| **96** | 211 | 1.346610 | 0.241262 | 0.984269 | 0 | 0.251499 | 0.095903 |
| **97** | 2212 | 0.993210 | 0.755494 | 0.733585 | 0 | 0.000000 | 0.000000 |
| **98** | 211 | 0.596323 | 0.722834 | 0.971532 | 0 | 0.000000 | 0.000000 |
| **99** | 2212 | 2.490730 | 0.210577 | 0.926961 | 0 | 0.033097 | 0.515953 |

100 rows × 7 columns

In [5]:
```
## positron (-11), pion (211), kaon (321), and proton (2212)
np.unique(file_data.id)
```

Out[5]:
```
array([ -11,  211,  321, 2212], dtype=int64)
```

In [6]:
```
# plt.hist(file_data.id)
```

In [7]:
```
### Are zero values, bad data points ?
### Should we remove those points?
# for icolumn in output_columns.columns.tolist():
#     output_column.icolumn[output_column.icolumn != 0]
```

In [8]:
```
# file_data.nphe[file_data.nphe != 0]
```

In [9]:
```
# file_data.ein[file_data.ein != 0]
```

In [10]:
```
file_data_dropna = file_data.dropna()
file_data_dropna.id[ file_data_dropna.id == 211] = 1
file_data_dropna.id[ file_data_dropna.id != 1] = 0
```

```
C:\Users\sarth\Miniconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\sarth\Miniconda3\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/u
ser_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [11]:
```
file_data_dropna
```

Out[11]:

| | id | p | theta | beta | nphe | ein | eout |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 0.780041 | 1.081480 | 0.989962 | 0 | 0.000000 | 0.000000 |
| **1** | 1 | 0.260929 | 0.778892 | 0.902450 | 0 | 0.000000 | 0.000000 |
| **2** | 0 | 0.773022 | 0.185953 | 0.642428 | 4 | 0.101900 | 0.000000 |
| **3** | 1 | 0.476997 | 0.445561 | 0.951471 | 0 | 0.000000 | 0.000000 |
| **4** | 0 | 2.123290 | 0.337332 | 0.908652 | 0 | 0.034379 | 0.049256 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **4999995** | 1 | 0.835889 | 0.495847 | 0.975812 | 0 | 0.000000 | 0.046967 |
| **4999996** | 1 | 2.027470 | 0.287966 | 1.222890 | 0 | 0.197894 | 0.186404 |
| **4999997** | 1 | 0.827497 | 0.689746 | 0.980957 | 0 | 0.000000 | 0.000000 |
| **4999998** | 0 | 1.331200 | 0.382746 | 0.811818 | 0 | 0.036942 | 0.056947 |
| **4999999** | 0 | 2.956890 | 0.449482 | 0.946111 | 0 | 0.106844 | 0.165438 |

5000000 rows × 7 columns

In [12]:
```
file_data_dropid = file_data_dropna.drop('id', axis =1)
particle_id = file_data_dropna.id
type([particle_id])
```

Out[12]:
```
list
```

In [13]:
```
len(file_data_dropna.id[file_data_dropna.id == 1])
```

Out[13]:
```
2806833
```

In [14]:
```
##  x --> id
## y --> all other data columns

x_train,x_test,y_train,y_test=train_test_split(file_data_dropid, particle_id,test_si
# x_train,x_test,y_train,y_test=train_test_split(file_data.id, file_data.drop('id',
```

In [15]:
```
x_train
```

Out[15]:

| | p | theta | beta | nphe | ein | eout |
|---|---|---|---|---|---|---|
| **2242983** | 1.263600 | 0.508638 | 0.794777 | 0 | 0.018860 | 0.092195 |
| **4734220** | 0.325780 | 0.658881 | 0.916117 | 0 | 0.000000 | 0.000000 |
| **2173219** | 1.401850 | 0.164202 | 0.828889 | 0 | 0.350103 | 0.129824 |
| **4534156** | 0.527810 | 0.202551 | 0.953051 | 0 | 0.037812 | 0.097780 |
| **3731796** | 0.384985 | 1.240200 | 0.926944 | 0 | 0.000000 | 0.000000 |
| **...** | ... | ... | ... | ... | ... | ... |
| **2219731** | 1.864810 | 0.244980 | 0.994369 | 0 | 0.165164 | 0.057038 |
| **2249467** | 2.276950 | 0.193339 | 0.999086 | 0 | 0.057359 | 0.054154 |
| **2215104** | 1.082020 | 0.506312 | 0.784595 | 0 | 0.028565 | 0.062440 |
| **1484405** | 0.896773 | 0.224454 | 0.685746 | 0 | 0.090593 | 0.000000 |
| **4500015** | 1.124130 | 0.326757 | 0.770327 | 0 | 0.115358 | 0.214557 |

3750000 rows × 6 columns

In [16]:
```
## Classification

from sklearn.ensemble import RandomForestClassifier
RFC=RandomForestClassifier()
RFC=RandomForestClassifier( n_estimators=200, criterion='entropy', random_state=0, n
```

In [17]:
```
RFC.fit(x_train, y_train)
```

Out[17]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=-1, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

In [18]:
```
RFC_pred=RFC.predict(x_test)
```

In [19]:
```
from sklearn.metrics import accuracy_score
```

In [20]:
```
RFC_accuracy=accuracy_score(RFC_pred,y_test)
print(RFC_accuracy)
```

```
0.975948
```

In [21]:
```
print("\x1b[31m", RFC_accuracy, "\x1b[0m")
```

```
0.975948
```

## **** Trying different combinations of parameters ****

In [22]:
```
#**** another  Model ******

RFC_2 =RandomForestClassifier( n_estimators=200, criterion='gini', random_state=0, n

RFC_2.fit(x_train, y_train)

RFC_pred_2=RFC_2.predict(x_test)

print(accuracy_score(RFC_pred_2,y_test))
```

```
0.9759064
```

In [23]:
```
#**** another  Model ******
RFC_3 = RandomForestClassifier( n_estimators=100, criterion='entropy', random_state=
RFC_3.fit(x_train, y_train)
RFC_pred_3=RFC_3.predict(x_test)
print(accuracy_score(RFC_pred_3,y_test))
```

```
0.9758728
```

In [24]:
```
#**** another  Model ******
RFC_4 = RandomForestClassifier( n_estimators=300, criterion='entropy', random_state=
RFC_4.fit(x_train, y_train)
RFC_pred_4=RFC_4.predict(x_test)
print(accuracy_score(RFC_pred_4,y_test))
```

```
0.9760776
```

In [25]:
```
#**** another  Model ******
RFC_5 = RandomForestClassifier( n_estimators=100, criterion='gini', random_state=0,
RFC_5.fit(x_train, y_train)
RFC_pred_5=RFC_5.predict(x_test)
print(accuracy_score(RFC_pred_5,y_test))
```

```
0.9758432
```

In [26]:
```
#**** another  Model ******
RFC_6 = RandomForestClassifier( n_estimators=100, criterion='entropy', bootstrap = T
RFC_6.fit(x_train, y_train)
RFC_pred_6=RFC_6.predict(x_test)
print(accuracy_score(RFC_pred_6,y_test))
```

```
0.9758728
```

## **** Learning Curve ****

In [41]:
```
file_data_dropid.shape
# particle_id.shape
indices = np.arange(particle_id.shape[0])

np.random.shuffle(indices)

X, y = file_data_dropid.iloc[indices], particle_id[indices]
```

In [42]:
```
from sklearn.model_selection import learning_curve
train_sizes, train_scores, valid_scores = learning_curve( RFC
    , X, y, train_sizes=[0.10, 0.25, 0.50], cv=3)
```

In [44]:
```
train_scores
```

Out[44]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [45]:
```
valid_scores
```

Out[45]:
```
array([[0.9752974 , 0.975262  , 0.97524159],
       [0.9756526 , 0.975694  , 0.97545279],
       [0.9758806 , 0.975823  , 0.97560519]])
```

In [ ]:

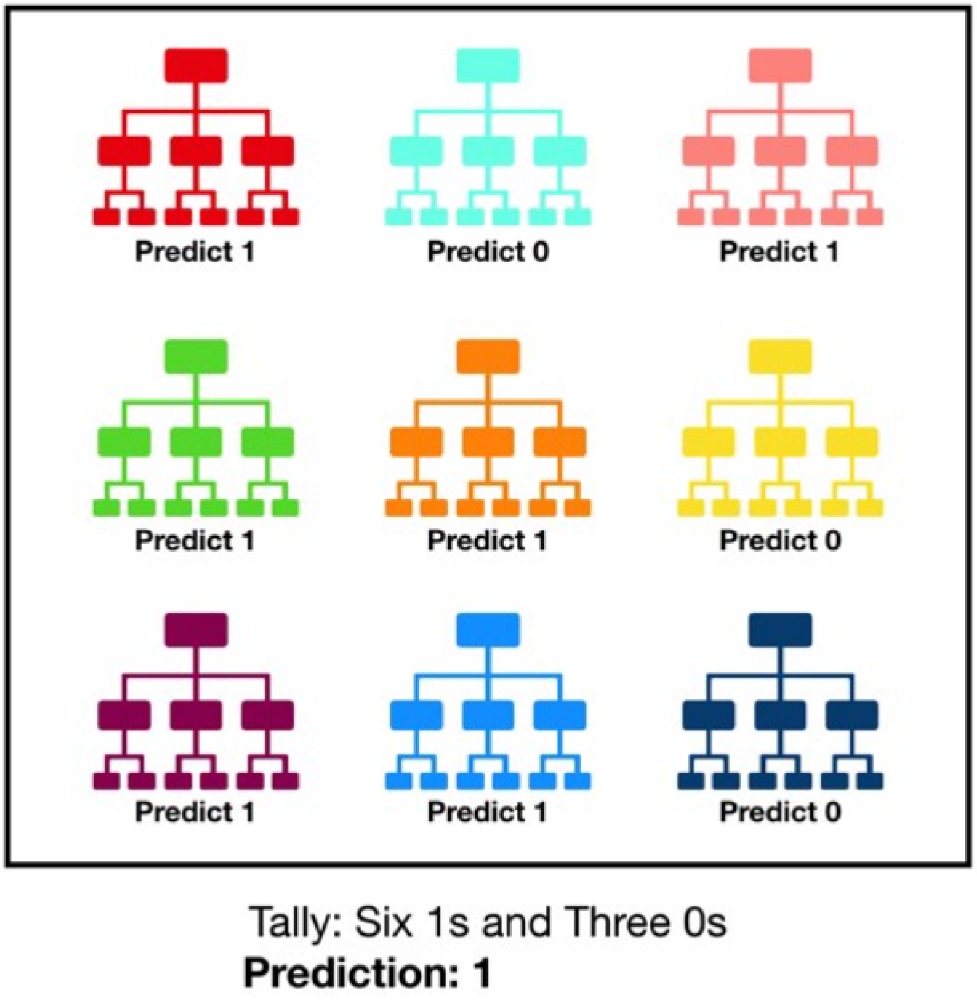


Figure 2.1: Random Forest Visualization, source: towardsdatascience.com

Chapter 3

# Conclusion

We have built several Random Forest models with different parameters. For all the different combinations of parameters the accuracy of model doesn't change significantly.
By applying 'curve_learning' function, we tested how the model behaves with different training data sets. We observe that both the training scores as well as validation scores are consistent as the size of training data set is increased from 10% to 50% of the full data set.
*Hence, it can be concluded that Random Forest based model is able to identify Pions with high accuracy.*

## Future Tasks:

- The accuracy of this approach needs to be verified on experimental data.
- As the fitting with Random Forest is quite time consuming, performance of the algorithm needs to be optimized, probably by converting ML model into tensor computations with help of python libraries such as HummingBird.
- To explore possibility of using Machine Learning techniques to identify *hard* to detect particles such as dark matter candidate Weakly Interacting Massive Particles (WIMP).