A Brief report of what I did the past week

Kostas Papadimos

Contents

1 The problem

There are two data sets(groups). Each data set consist of some sets of 4 numbers. The sum of the numbers in each set of a group is the same. The goal is to train a boosted decision tree such that it will be able to separate the numbers of each group.

2 Method

For simplicity, I was working with integer numbers. Moreover I created the the testing and training data sets in 3 different ways.

- I selected the number that every set of a given group will add up to
- The sum of every set in a given group was selected randomly
- The sum of every set in a given group was not the same. The numbers of each set summed to a number coming from a normal distribution

2.1 Creating the data set

2.1.1 Fixed sum

Having the number that each set of each group will add up to, I picked 3 random numbers in the range [-100, 100] and subtracted their sum from the fixed number. This way I created two groups of 10000 sets each.

```
def rand_data(s, n):
  '''
  returns n numbers, n1, n2, ... such that n1+n2+...=s
  '''
  import random as rand
  nums = [rand.randint(-100, 100) for i in range(n-1)]
  nums.append(s - sum(nums))
  return nums
```

```
#
s1, s2 = (2, 9) # randomly picked sum
n = 4 # numbers in each set
n_sets = 10000 # number of sets in each group
#
group_a = [rand_data(s, n) for i in range(n_sets)]
group_b = [rand_data(s, n) for i in range(n_sets)]
```

In the histogram bellow, it is verified that the stets in each group add up to the same number.

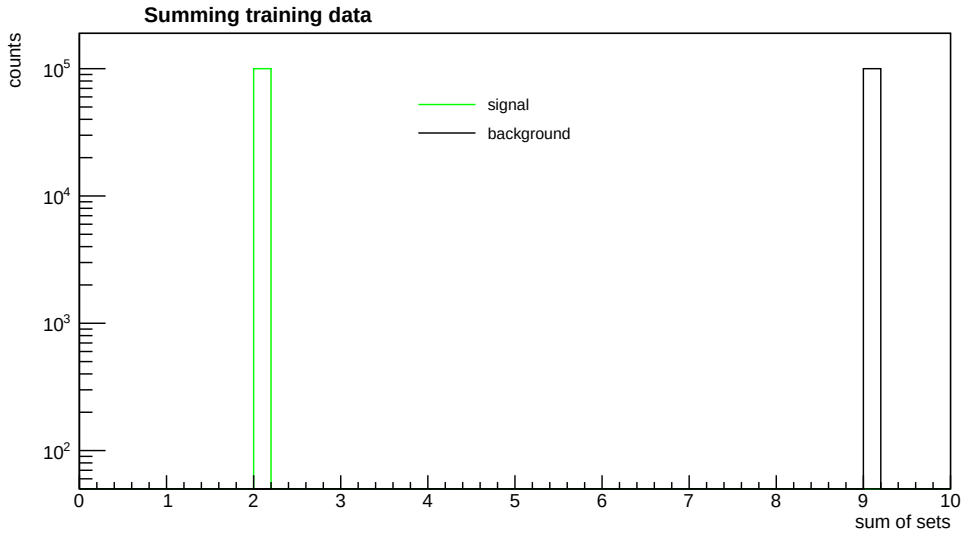

Figure 1: sum of each group. Using fixed sums

Obviously the sums of each group are s = 2 and s = 9.

2.1.2 Random sum

The number that each set of each group will add up to, is not fixed by me, but its randomly selected in the range[0, 10], using a random number generator(python). The way of creating the numbers of each set is the same as before:

```
import random
s1, s2 = [random.randint(0, 10) for i in range(num_groups)] # randomly generated sum
n = 4 # numbers in each set
n_sets = 10000 # number of sets in each group
#
group_a = [rand_data(s, n) for i in range(n_sets)]
group_b = [rand_data(s, n) for i in range(n_sets)]
```

In the histogram bellow, it is verified that the sets in each group add up to the same number. Obviously the sums of each group are s = 5 and s = 8.

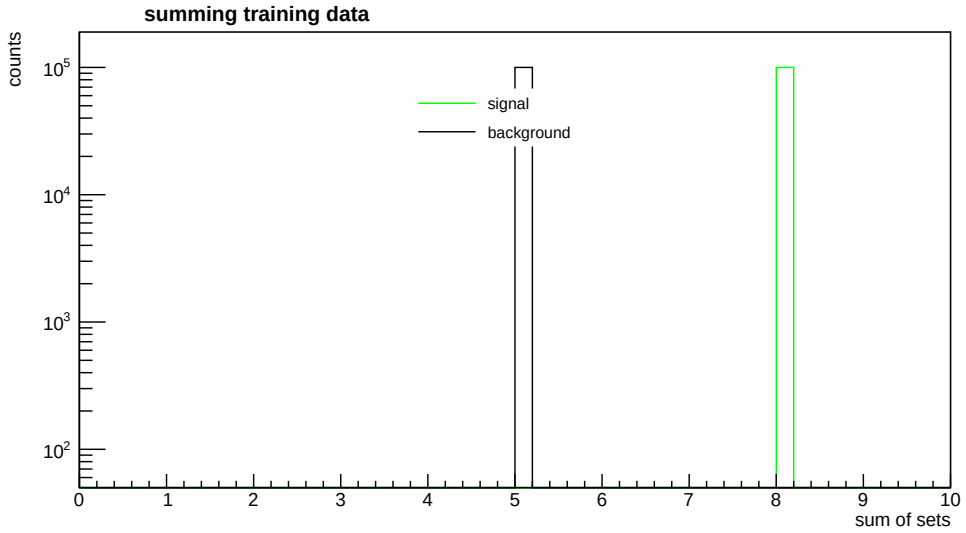

Figure 2: sum of each group. Using random sums

2.1.3 Normally distributed sum

The sets of a given group do not add up to the same number. Instead, the number that each set adds up to, follows the Gaussian distribution with fixed means and sigmas for each group:

```
mu_a, mu_b = (2, 9) # fixed means
sigma = 0.1 # for simplicity the sigma is kept the same
n_sets = 10000 # number of sets in each group
#
group_a = [rand_data(rand.gauss(mu_a, sigma), n) for i in range(n_sets)]
group_b = [rand_data(rand.gauss(mu_a, sigma), n) for i in range(n_sets)]
```

In the histogram bellow, it is verified that the sets of a given group add up to a number thats normally distributed. The means of each distribution are $\mu = 2$ and $\mu = 9$

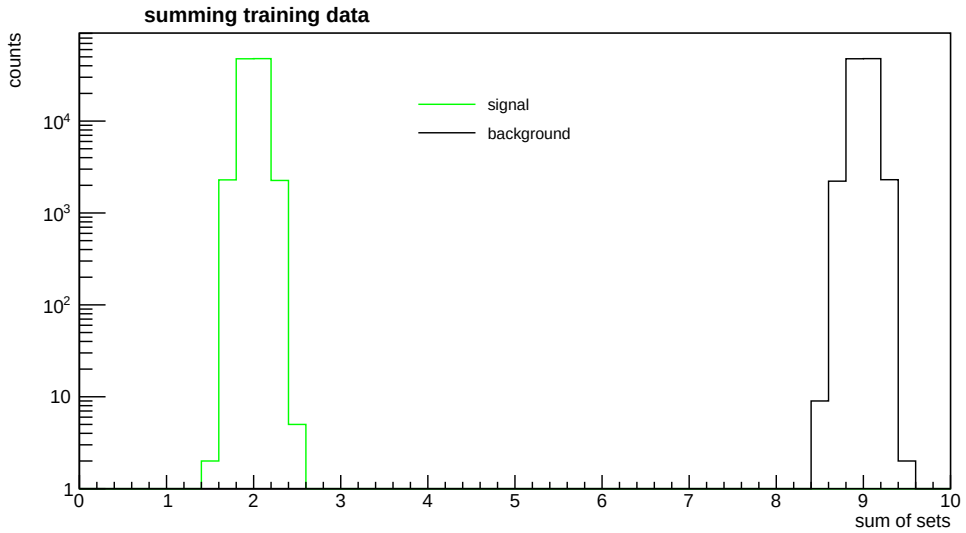

Figure 3: sum of each group. Using normally distributed sums

2.2 Testing and Training

Both training and testing was done exactly as shown in the example at `https://root.cern.ch/doc/master/tmva101__Training_8py.html`.

3 Results

The ROC curves as well as the AUC scores of each model are shown in the plot bellow:

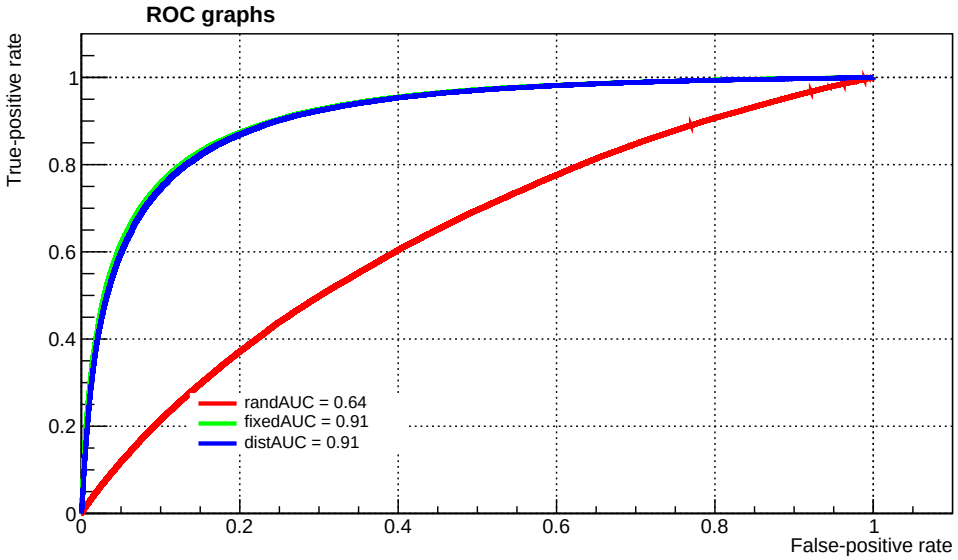

Figure 4: Efficiency of each model based on the way that the data was generated