

# Lab 1 Report:

## Data Preparation Techniques for Machine Learning

Name: Forest Tschirhart

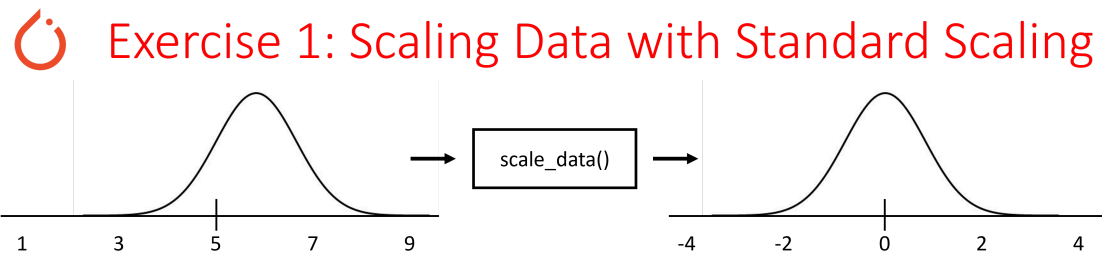
In [1]: *# Import necessary libraries*

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]: *from IPython.display import Image # For displaying images in colab jupyter c*

In [3]: `Image('lab1_exercisel.png', width = 1000)`

Out[3]:



- In Machine Learning, the dataset is usually scaled ahead of time so that it is easier for the computer to **learn** and **understand** the problem.
- One of the most frequently used method is 'standard scaling', where the data is scaled by  $z = (x - \mu)/\sigma$ . ( $x$  = original datapoint,  $\mu$  = mean of the data,  $\sigma$  = standard deviation)
- Write a function "scale\_data()" which takes 2D NumPy array as an input and perform standard scaling on its columns. The function should output a new 2D array containing scaled column data.
- Test your function with selected columns in CMS calorimeter dataset (`hgcal.csv`).
- Plot the scaled dataset for the selected columns by using the provided matplotlib histogram function.

66

In [4]: *# Load the dataset (.csv) using pandas package*

```
CMS_calori_dataset = pd.read_csv('hgcal.csv')

# .head directive on the panda dataframe displays the first n-rows

CMS_calori_dataset.head(n = 10)
```

Out[4]:

	Unnamed: 0	x	y	z	eta	phi	energy	trackId
0	0	179.50383	-23.632137	-7.878280	-0.0435	-0.130900	0.200126	462412
1	1	-143.63881	110.217940	-72.706795	-0.3915	2.487094	2.734594	493395
2	2	179.50383	-23.632120	-146.429610	-0.7395	-0.130900	0.423910	1
3	3	-172.67310	54.443620	-238.065340	-1.0875	2.836160	0.713950	493640
4	4	-180.88046	7.897389	-238.065340	-1.0875	3.097959	0.000000	495225
5	5	-180.88045	-7.897438	-238.065340	-1.0875	-3.097959	0.034491	495225
6	6	-152.69838	-97.279590	-265.020540	-1.1745	-2.574361	0.580138	460126
7	7	-23.63213	179.503810	-325.172060	-1.3485	1.701696	0.411487	465028
8	8	-152.69835	97.279594	89.977780	0.4785	2.574361	0.183141	1383
9	9	-176.76110	39.187016	107.930240	0.5655	2.923426	0.337551	4421

In [5]: *# Convert the panda dataframe into numpy 2D array*

```

CMS_calori_dataset_np = CMS_calori_dataset.to_numpy()
#print(CMS_calori_dataset_np)

# The converted numpy array has the dimension of 420 (rows) x 8 (columns)

print(CMS_calori_dataset_np.shape)

```

(420, 8)

In [6]: *# Extract only x, y, z, eta, phi and energy columns from the dataset and store it in a new 2D array CMS\_calori\_dataset\_np\_sub.  
# Name this new 2D array CMS\_calori\_dataset\_np\_sub.  
# The array should have dimension 420 (rows) x 6 (columns)*

```

CMS_calori_dataset_np_sub = CMS_calori_dataset_np[:,1:-1] # data extraction
print(CMS_calori_dataset_np_sub.shape)

```

(420, 6)

In [7]: *# Create the scaling function  
# this does the assigned mathematical function elementwise on arr.  
# However it expects a specific array shape and will break if given wrong shape*

```

def scale_data(arr):
    scaled_data = [] # init empty list
    dim = arr.shape[1] # get the number of columns
    for i in range(dim): # for each data type (x, y, z, etc.) apply respective scaling
        mean = np.mean(arr[:, i])
        std = np.std(arr[:, i])
        temp_arr = (arr[:, i] - mean) / std
        scaled_data.append(temp_arr) # fill empty list
    scaled_data = np.array(scaled_data) # convert list to array
    return np.transpose(scaled_data) # transpose the array to get the correct shape

```

In [8]: *# Test the function with CMS\_calori\_dataset\_np\_sub*

```
CMS_calori_dataset_np_sub_scaled = scale_data(CMS_calori_dataset_np_sub)
print(CMS_calori_dataset_np_sub_scaled[0])
```

```
[ 1.91214438 -0.51027049 -0.44193343 -0.47341363 -0.31488841 -0.38410307]
```

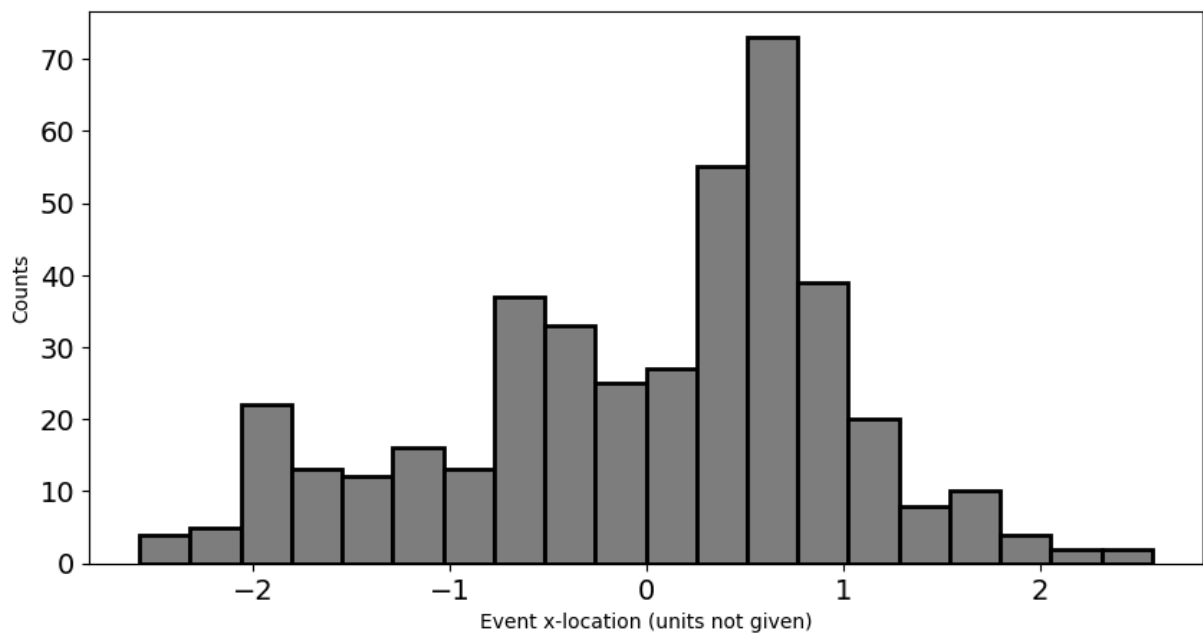
In [9]: *# Confirm the data is scaled for 'x' column*

```
plt.figure(figsize = (10, 5))

plt.hist(CMS_calori_dataset_np_sub_scaled[:, 0], bins = 20, facecolor = 'green')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.xlabel('Event x-location (units not given)') # Units not given in assignment
plt.ylabel('Counts')

plt.show()
```



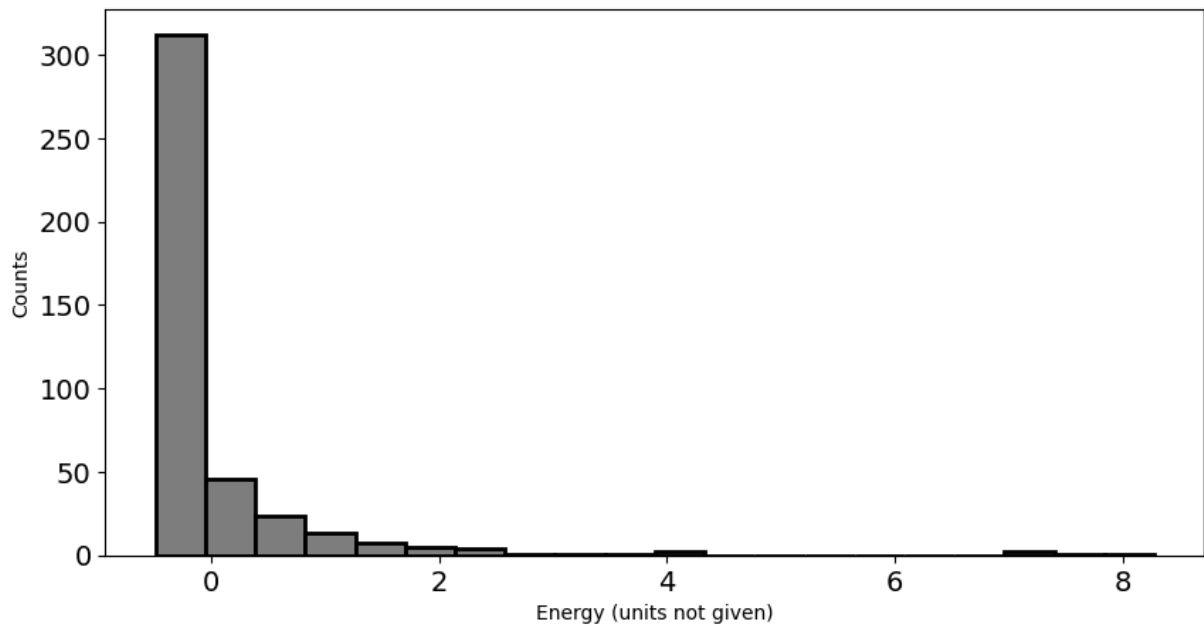
In [10]: *# Confirm the data is scaled for 'energy' column*

```
plt.figure(figsize = (10, 5))

plt.hist(CMS_calori_dataset_np_sub_scaled[:, 5], bins = 20, facecolor = 'green')
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.xlabel('Energy (units not given)')
plt.ylabel('Counts')

plt.show()
```

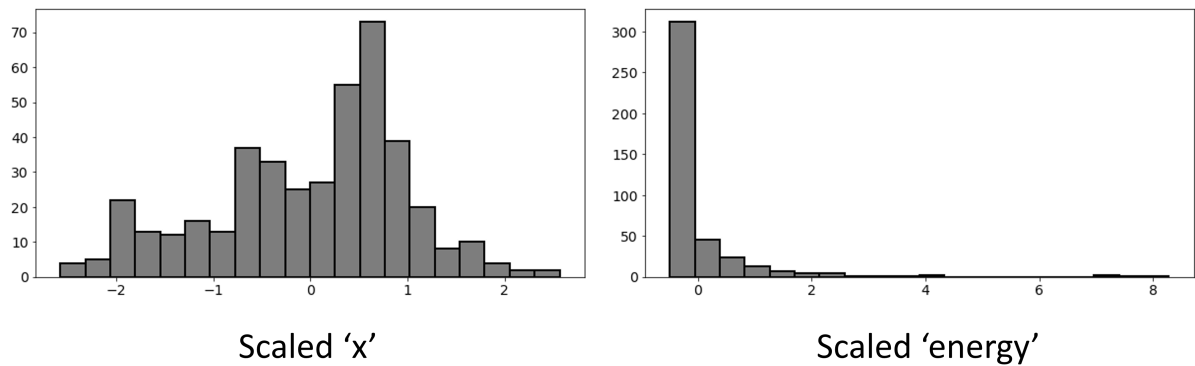


**Expected histogram outputs - Feel free to style your plot differently**

In [11]: *# Both histograms agree with my data manipulation*

`Image('lab1_e1_expected_outputs.png', width = 1000)`

Out[11]:

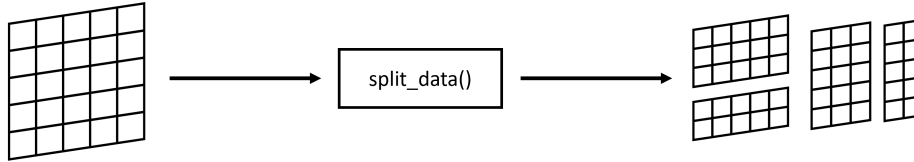


In [12]: `Image('lab1_exercise2.png', width = 1000)`

Out[12]:



## Exercise 2: Data Splitting



- In this exercise you will write a function called `split_data()` which given a NumPy array, it splits the array into sub-arrays.
- Data splitting is used to divide the dataset into training, validation and testing sets, which we will describe in later lab.
- The function should take following parameters
  - `arr` – 2D NumPy array representing a dataset
  - `split_proportions` – a list containing split ratios, e.g., `[0.2, 0.3, 0.5]`
  - `axis` – a direction to be splitted (0 = row-wise, 1 = column-wise)
- Test your function on the scaled dataset from exercise 1 with given parameters in the lab template.
- Confirm that your sub arrays have correct dimensions by printing their shape

68

```
In [13]: # Create the splitting function
# Wrote this before I knew about np.split
def split_data(arr, split_proportions, axis):

    if not np.isclose(np.sum(split_proportions), 1): # make sure split proportions sum to 1
        raise ValueError("Try again. The sum of split proportions must be equal to 1")

    # normalize proportions to array size. Convert to int for indexing
    slice_sizes = (np.array(split_proportions) * arr.shape[axis]).astype(int)
    slice_sizes = slice_sizes[:-1] # last element not needed in loop

    split_data_list = [] # init empty list to be filled by for loop
    for i in slice_sizes:
        if axis == 0: # make sure slicing syntax is correct for desired axis
            temp_arr = arr[:i,:] # grab specified portion
            arr = arr[i:,:] # remove the grabbed portion from the original array so that next iteration can also start from beginning
        else:
            temp_arr = arr[:, :i]
            arr = arr[:, i:]
        split_data_list.append(temp_arr) # fill the list with the sliced data

    split_data_list.append(arr) # append the leftover portion to the split data list

    return split_data_list

# This is a better way of doing the same thing but I kept the above function for reference
def split_data_numpy(arr, split_proportions, axis):

    # normalize proportions to array size. Convert to int for indexing
    if not np.isclose(np.sum(split_proportions), 1):
        raise ValueError("Try again. The sum of split proportions must be equal to 1")

    # Convert proportions to index numbers of the source array. eg. [0.2, 0.3, 0.5] becomes [10, 15, 25] for a 50x50 array
    indices = (np.cumsum(np.array(split_proportions)) * arr.shape[axis]).astype(int)
```

```
indices = indices[:-1] # last element not needed
split_data_list = np.split(arr, indices, axis=axis) # nice and easy numpy
return split_data_list
```

```
In [14]: # Test your split function against scaled CMS Calorimeter dataset from exer
sub_data_list_1 = split_data(arr = CMS_calori_dataset_np_sub_scaled,
                             split_proportions = [0.6, 0.2, 0.2], axi
```

```
In [15]: # Confirm that dataset has been split into correct shapes
# The correct dimensions should be (252, 6) (84, 6) (84, 6)

print(sub_data_list_1[0].shape, sub_data_list_1[1].shape, sub_data_list_1[2]

(252, 6) (84, 6) (84, 6)
```

```
In [16]: # Test your split function against scaled CMS Calorimeter dataset from exer
sub_data_list_2 = split_data(arr = CMS_calori_dataset_np_sub_scaled,
                             split_proportions = [0.5, 0.
```

```
In [17]: # Confirm that dataset has been split into correct shapes
# The correct dimensions should be (420, 3) (420, 3)

print(sub_data_list_2[0].shape, sub_data_list_2[1].shape)

(420, 3) (420, 3)
```

```
In [ ]:
```

```
In [ ]:
```