SFT Group Meeting

Status Update #11

Konrad Helms

7th September 2023

Recent Activity

- generated a new dataset
- fixed bugs in data split
- retrained the model
- evaluated the model performance

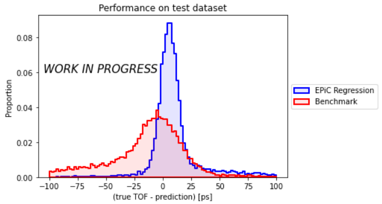
- in the following: 36257 pfos for training, 14523 pfos for testing

Model Performance - after Epoch 14

```
# GETTING RMS90
mean90, rms90, mean90_err, rms90_err = fit90(test_diff[abs(test_diff) <= 100])
print('ML RMS90', rms90, '+-', rms90_err)
mean90, rms90, mean90_err, rms90_err = fit90(classical_algorithm[abs(classical_algorithm) <= 100])
print('CLASSICAL RMS90', rms90,  '+-', rms90_err)
```

```
ML RMS90 9.705123116791874 +- 0.06191284995396701
CLASSICAL RMS90 23.146844320676372 +- 0.1521830994976098
```

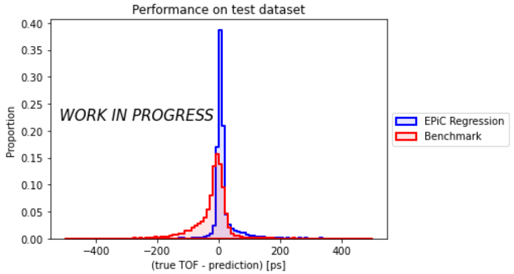
•••



```
# GETTING RMS90
mean90, rms90, mean90_err, rms90_err = fit90(test_diff)
print('ML RMS90', rms90, '+-', rms90_err)
mean90, rms90, mean90_err, rms90_err = fit90(classical_algorithm)
print('CLASSICAL RMS90', rms90, '+-', rms90_err)
```

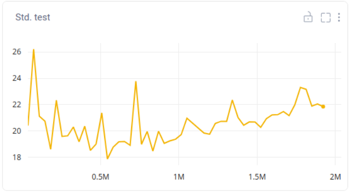
```
ML RMS90 14.953687860893536 +- 0.09248661953543823
CLASSICAL RMS90 32.760697579923985 +- 0.20380915751853218
```

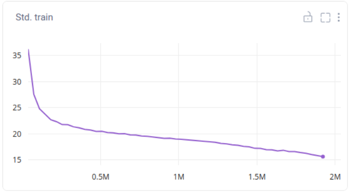
•••

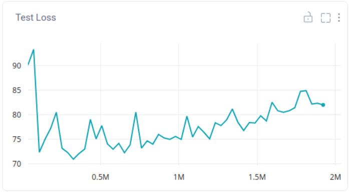


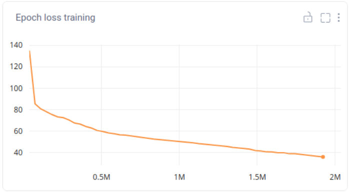
Sanity Check #1 - see Overtraining - Good!

1 Epoch = 36257 steps



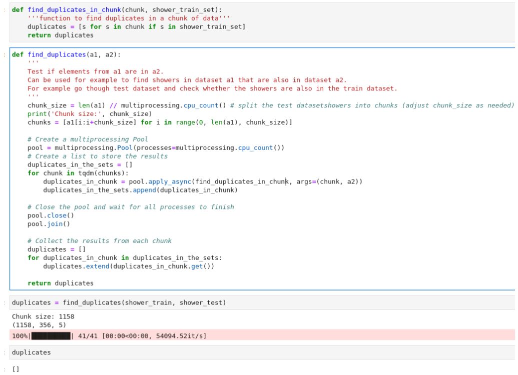






Sanity Check #2 - All Good!

Check if test showers are in training dataset (individually!)



```
def find_duplicates_in_chunk(chunk, shower_train_set):
    '''function to find duplicates in a chunk of data'''
    duplicates = [s for s in chunk if s in shower_train_set]
    return duplicates
```

```
def find_duplicates(a1, a2):
    '''
    Test if elements from a1 are in a2.
    Can be used for example to find showers in dataset a1 that are also in dataset a2.
    For example go though test dataset and check whether the showers are also in the train dataset.
    '''
    chunk_size = len(a1) // multiprocessing.cpu_count() # split the test datasetshowers into chunks (adjust chunk_size as needed)
    print('Chunk size:', chunk_size)
    chunks = [a1[i:i+chunk_size] for i in range(0, len(a1), chunk_size)]

    # Create a multiprocessing Pool
    pool = multiprocessing.Pool(processes=multiprocessing.cpu_count())
    # Create a list to store the results
    duplicates_in_the_sets = []
    for chunk in tqdm(chunks):
        duplicates_in_chunk = pool.apply_async(find_duplicates_in_chunk, args=(chunk, a2))
        duplicates_in_the_sets.append(duplicates_in_chunk)

    # Close the pool and wait for all processes to finish
    pool.close()
    pool.join()

    # Collect the results from each chunk
    duplicates = []
    for duplicates_in_chunk in duplicates_in_the_sets:
        duplicates.extend(duplicates_in_chunk.get())

    return duplicates
```

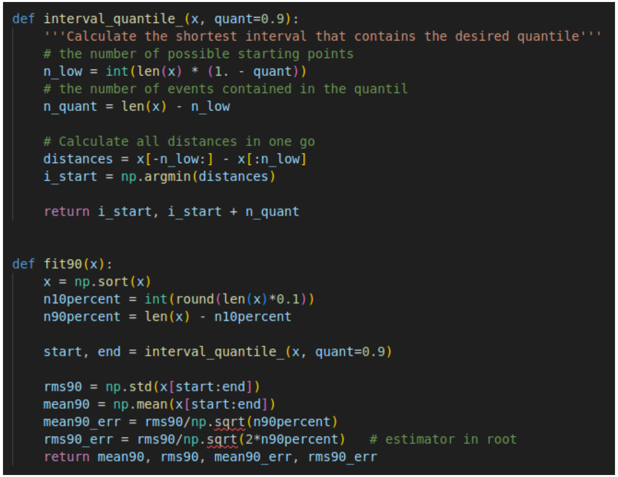
```
duplicates = find_duplicates(shower_train, shower_test)
```

```
Chunk size: 1158
(1158, 356, 5)
100%|██████████| 41/41 [00:00<00:00, 54094.52it/s]
```

```
duplicates
```

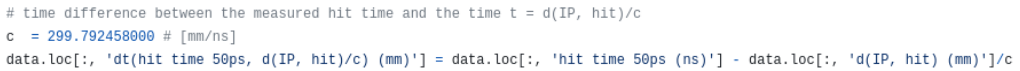
```
[]
```

RMS90

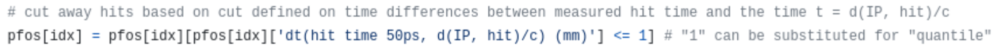


```
def interval_quantile_(x, quant=0.9):
    '''Calculate the shortest interval that contains the desired quantile'''
    # the number of possible starting points
    n_low = int(len(x) * (1. - quant))
    # the number of events contained in the quantil
    n_quant = len(x) - n_low

    # Calculate all distances in one go
    distances = x[-n_low:] - x[:n_low]
    i_start = np.argmin(distances)

    return i_start, i_start + n_quant


def fit90(x):
    x = np.sort(x)
    n10percent = int(round(len(x)*0.1))
    n90percent = len(x) - n10percent

    start, end = interval_quantile_(x, quant=0.9)

    rms90 = np.std(x[start:end])
    mean90 = np.mean(x[start:end])
    mean90_err = rms90/np.sqrt(n90percent)
    rms90_err = rms90/np.sqrt(2*n90percent)   # estimator in root
    return mean90, rms90, mean90_err, rms90_err
```

Data Preprocessing - Things I want to Point Out



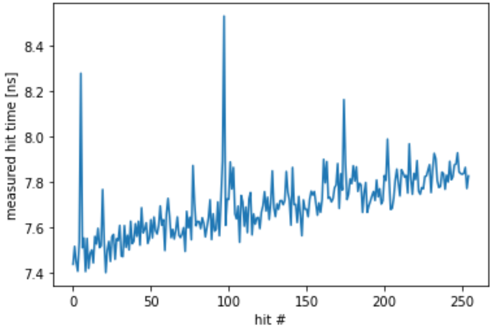
```
# time difference between the measured hit time and the time t = d(IP, hit)/c
c  = 299.792458000 # [mm/ns]
data.loc[:, 'dt(hit time 50ps, d(IP, hit)/c) (mm)'] = data.loc[:, 'hit time 50ps (ns)'] - data.loc[:, 'd(IP, hit) (mm)']/c
```



```
# cut away hits based on cut defined on time differences between measured hit time and the time t = d(IP, hit)/c
pfos[idx] = pfos[idx][pfos[idx]['dt(hit time 50ps, d(IP, hit)/c) (mm)'] <= 1] # "1" can be substituted for "quantile"
```

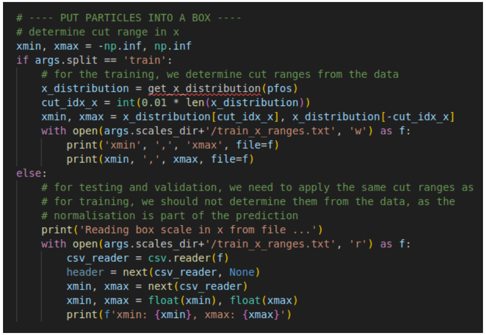
remark: 78% of the data have dt(hit time 50 ps, d(IP, hit)/c) $\leq$ 1 ns -> we still see some outliers 'by eye' but got rid of the very large outliers

Data Preprocessing - Outliers



Data Preprocessing - Things I want to Point Out

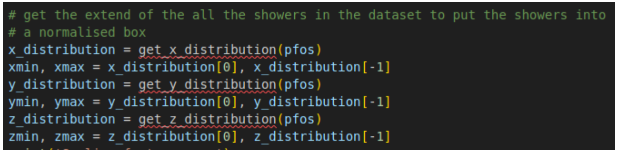
Using the training dataset, determine the box size for the particles to be put into (for x,y,z)



```python
# ---- PUT PARTICLES INTO A BOX ----
# determine cut range in x
xmin, xmax = -np.inf, np.inf
if args.split == 'train':
    # for the training, we determine cut ranges from the data
    x_distribution = get_x_distribution(pfos)
    cut_idx_x = int(0.01 * len(x_distribution))
    xmin, xmax = x_distribution[cut_idx_x], x_distribution[-cut_idx_x]
    with open(args.scales_dir+'/train_x_ranges.txt', 'w') as f:
        print('xmin', ',', 'xmax', file=f)
        print(xmin, ',', xmax, file=f)
else:
    # for testing and validation, we need to apply the same cut ranges as
    # for training, we should not determine them from the data, as the
    # normalisation is part of the prediction
    print('Reading box scale in x from file ...')
    with open(args.scales_dir+'/train_x_ranges.txt', 'r') as f:
        csv_reader = csv.reader(f)
        header = next(csv_reader, None)
        xmin, xmax = next(csv_reader)
        xmin, xmax = float(xmin), float(xmax)
        print(f'xmin: {xmin}, xmax: {xmax}')
```

Data Preprocessing - Things I want to Point Out

Then, we apply the cuts, i.e. throw away hits outside of the box.
Lastly, we determine the extend of the showers. For example: take **all** the x values from **all** hits in the dataset and determine min and max. Set min=-1, max=1, scale accordingly. Same for y and z.
This we do for the training, testing and validation dataset.!



```
# get the extend of the all the showers in the dataset to put the showers into
# a normalised box
x_distribution = get_x_distribution(pfos)
xmin, xmax = x_distribution[0], x_distribution[-1]
y_distribution = get_y_distribution(pfos)
ymin, ymax = y_distribution[0], y_distribution[-1]
z_distribution = get_z_distribution(pfos)
zmin, zmax = z_distribution[0], z_distribution[-1]
```