Model trainer and porter

Obviously, we start by importing the required packages.

```
from everywhereml.sklearn.ensemble import RandomForestClassifier
import numpy as np
from everywhereml.data import Dataset
import pandas as pd
print(pd. __version__)
```

First the script reads all the collected data from text files and writes them into a NumPy arrays.

```
x_sine = np.loadtxt("sine_4.txt",
                    delimiter=",", dtype=np.float16, encoding="utf8")
x_square = np.loadtxt("sq_4.txt",
                    delimiter=",", dtype=np.float16, encoding="utf8")
x_saw = np.loadtxt("saw_4.txt",
                    delimiter=",", dtype=np.float16, encoding="utf8")
x_noise = np.loadtxt("noise_4.txt",
                    delimiter=",", dtype=np.float16, encoding="utf8")
```

We take note how many sets of samples each signals have and store this info into 4 different variables. This will be useful when labelling the data. Label in this context meaning marking all the sets of data with a corresponding integer (sine=0, saw=1, square=2, noise=3).

```
size_sine =  x_sine.shape
length_sin = size_sine[0]
size_sq =  x_square.shape
length_sq = size_sq[0]
size_saw =  x_saw.shape
length_saw = size_saw[0]
size_n =  x_noise.shape
length_n = size_n[0]
```

We use the length variables to create arrays filled with one of the label integers.

```
sin_labels = np.full((length_sin, 1), 0, np.float32)
sq_labels = np.full((length_sq, 1), 2, np.float32)
saw_labels = np.full((length_saw, 1), 1, np.float32)
noise_labels = np.full((length_n, 1), 3, np.float32)
```

The script then links all the data and the labels into 2 NumPy arrays.

```
combined_x = np.concatenate((x_sine, x_square, x_saw, x_noise), axis=0)
combined_y = np.concatenate((sin_labels, sq_labels, saw_labels, noise_labels),
axis=0)
```

It is now necessary to store the length of this newly created data array into an integer variable.

```
size_c =  combined_x.shape
length_c = size_c[0]
```

After that we convert any NaNs to 0 and create an empty array that will be one used for training. Note that this array has only 8 columns.

```
combined_x = np.nan_to_num(combined_x)
all = np.empty([length_c, 8])
```

We then create a char array with matching amount of space as sections of data.

```
size_y =  combined_y.shape
length_y = size_y[0]
all_names = [""] * length_y
```

The script creates a string array that can be used to name all the signals by using the correct index.

```
names_arr = np.array(["sine","saw","square","noise"])
```

With two nested for loops the script makes statistical analysis and stores it in the all array. Additionally, the labels are stored in the last column of the array.

```
for i in range(length_c):

        min = np.nanmin(combined_x[i,:])
        subtracted = combined_x[i,:] - min
        subtracted = np.nan_to_num(subtracted)
        max = np.nanmax(subtracted)
        divided = subtracted / max
        diff_1 = 0
        diff_5 = 0
        diff_20 = 0
        diff_50 = 0
        last1 = divided[0]
        last5 = divided[0]
        last20 = divided[0]
        last50 = divided[0]
        for k in range(254):
                diff_1 = diff_1 + abs(last1 - divided[k + 1])
                last1 = divided[k]

                if (k % 5 == 0):
                    diff_5 = diff_5 + abs(last5 - divided[k + 1])
                    last5 = divided[k]

                if (k % 10 == 0):
                        last20 = divided[k]
                        diff_20 = diff_20 + abs(last20 - divided[k + 1])
                if (k % 50 == 0):
                        last50 = divided[k]
                        diff_50 = divided[k] + abs(last50 - divided[k + 1])

        combined_x[i,:] = divided
        all[i,0] = diff_1 / 255
        all[i,1] = np.nanvar(combined_x[i,:])
        all[i,2] = np.nanstd(combined_x[i,:])
        all[i,3] = np.sqrt(np.nansum(np.square(combined_x[i,:])))
        all[i,4] = diff_5 / 50
        label = combined_y[i]
        all[i,5] = diff_20 / 12
        all[i,6] = diff_50 / 4
        all[i,7] = label

        all_names[i] = names_arr[int(label)]
```

The statistical analysis data is then converted from NumPy array to Pandas dataframe. This conversion is made so that it can be converted into a dataset which is a datatype defined by everywhereml libary. Also, the names of the categories are added to the dataframe.

```
columns_n =
("difference_1","var","std","sum","difference_5","difference_10","difference_50","t
arget")

all = np.nan_to_num(all[:,:])
df_f = pd.DataFrame(data = all,

                    columns=columns_n
                  )


df_f["names"] = all_names
print(df_f)
```

This dataframe is written into a csv file for validation/testing purposes.

```
df_f.to_csv('new.csv', index=False)
```

Then the conversion from dataframe to dataset is done.

```
ds = Dataset.from_pandas(df_f,"ds",target_name_column="names")
```

Now we can define the Random forest classifier with our wanted properties.

```
signal_classifier = RandomForestClassifier(n_estimators=20, max_depth=20)
```

The data needs to be split into two different sets for training and testing. test_size=0.2 means that 20% of the data is allocated for testing.

```
train, test = ds.split(test_size=0.2)
```

```
The model training is then done and after that the result are printed to console.
```

```
signal_classifier.fit(train)
print('Score on test set: %.2f' % signal_classifier.score(test))
```

The final part of the script ports the trained model to Arduino file so it can be used with Arduino sketch.

```
print(signal_classifier.to_arduino_file(
    'Classifier.h',
    instance_name='forest',
    class_map=ds.class_map
))
```