Data Science for Linac Controls (L-CAPE Project)

Brillina Wang

Elgin Community College

Supervisor: Jason St. John

Authors

Brillina Wang
Fermilab, Elgin Community College

Supervisor: Jason St. John
Fermilab

Introduction

Linear accelerators (Linacs) accelerate H- ions and subject them through a series of electric potentials along a linear beamline. They are mainly composed of radiofrequency (RF) cavities placed in-line with one another to provide a large amount of energy gain per unit length.

Artificial intelligence is a new and growing field involving machines that demonstrate intelligence instead of natural intelligence. Its applications are vast in modern technology and are often used in search engines, recommendation and filtering systems, self-driving cars, natural-language interpretation, automated decision-making, and strategic games. It can also contain elements from computer science, linguistics, psychology, and philosophy [1]. Machine learning (ML) refers to the training of machines to get better at a task without explicit programming. It is a field within AI that is focused on understanding and building methods to help improve performance on certain tasks. ML algorithms involve creating a model based on training data to make predictions and have been used in a large set of applications, including email filtering and speech recognition [2]. Anomaly detection is an ML method of identifying unexpected events or observations that differ significantly from the predicted value.

At Fermilab AD, the controls system for the accelerator complex monitors and issues commands to 4000+ control system parameters in the linear accelerator at frequencies ranging from 15 Hz to once every few minutes (Figure 1). Until the start of a beam interruption, this data is used by accelerator operators to investigate the source of the unplanned beam outage from the FNAL Main Control Room. The L-CAPE project is focused on predicting and preventing Linac beam outages with anomaly detection. L-CAPE (Linac Condition Anomaly Prediction of Emergence) is a project that aims to apply data science methods in order to improve information from the linear accelerator, to use machine learning techniques to automatically label unplanned beam outage types as they occur and to identify patterns within the data that could help predict and prevent the occurrence of future outages [3].

Introduction (cont.)

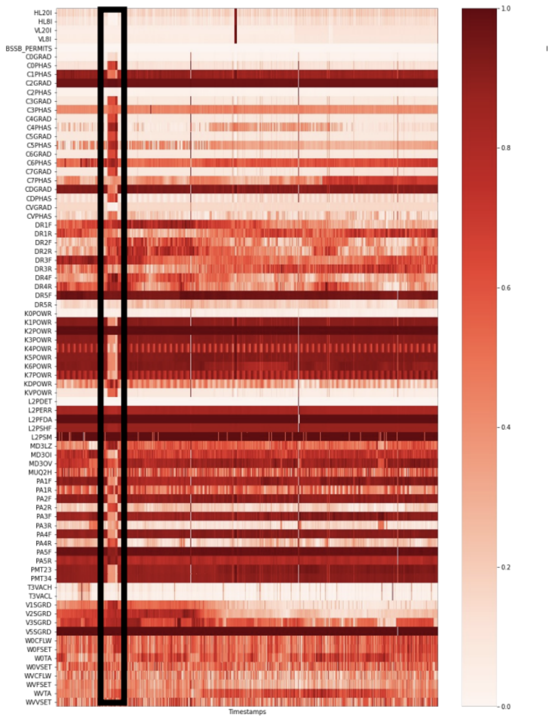


Figure 1: Heatmap of data collected by various devices (Jan Strube PNNL Private Correspondence)

Uniform Manifold Approximation and Projection (UMAP) is an algorithm that reduces the dimension of the data to be used for clustering. It would then capture patterns within a specific cluster and the general pattern between distinct clusters. This will be useful when identifying and categorizing outages as they arrive.

Purpose

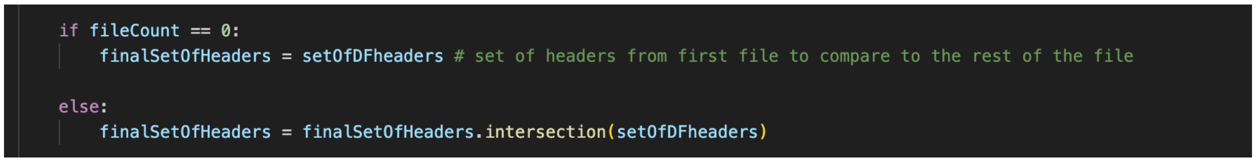
Training outage file data must be preprocessed and prepared before clustering. Then, the processed data is trained to identify and predict future beam outages on UMAP through unsupervised learning.

In this project that focuses on the first half, a data processing script was written to prepare the data so that it can be ready for training. Outage file data was read and processed in order to be prepared for the outage labeling step. These files contain data sequences around identified beam outages.

Methods

The set of Linac devices to use was found by looping over the outage files in the directory. The largest set of columns common to all files and the smallest number of rows before and after the outage onset were also found. The findings were documented so that they can be recalled when needed. The set of columns was reduced to only the RF station devices.

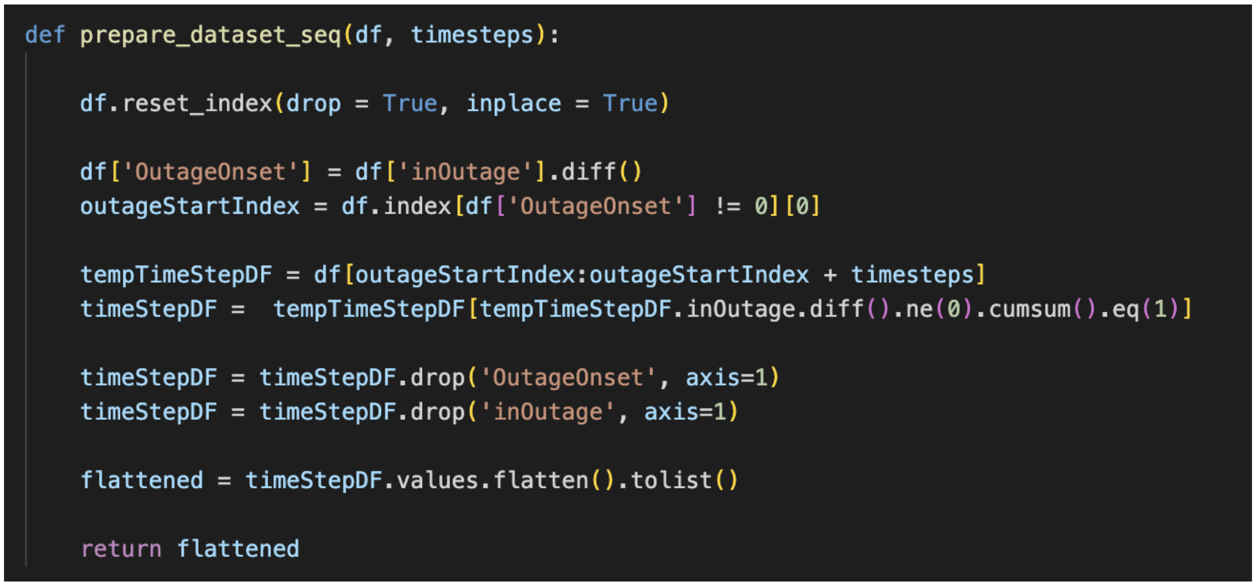
Each outage file of type .parquet.snappy was read in and converted to a Pandas DataFrame. In a loop for each file, a list of column names was created and each column name was added to a set. The function `.intersection()` was run on the set of column names for each file until the largest set of columns common to all files is obtained (Figure 2). Also, the smallest number of rows was found before and after the outage onset, or when the outage began, using the `min()` function for Lists.



```
if fileCount == 0:
    finalSetOfHeaders = setOfDFheaders # set of headers from first file to compare to the rest of the file

else:
    finalSetOfHeaders = finalSetOfHeaders.intersection(setOfDFheaders)
```

Figure 2: Finding the largest set of columns to all outage files

Two functions were then created and run both inside the loop. The first function, `prepare_dataset_seq`, takes the DataFrame created from the current outage file being read and a specified number of timesteps and returns a flattened list of data per every timestep until the outage starts. This process allows the data to be standardized for the final DataFrame, which will be used for UMAP.



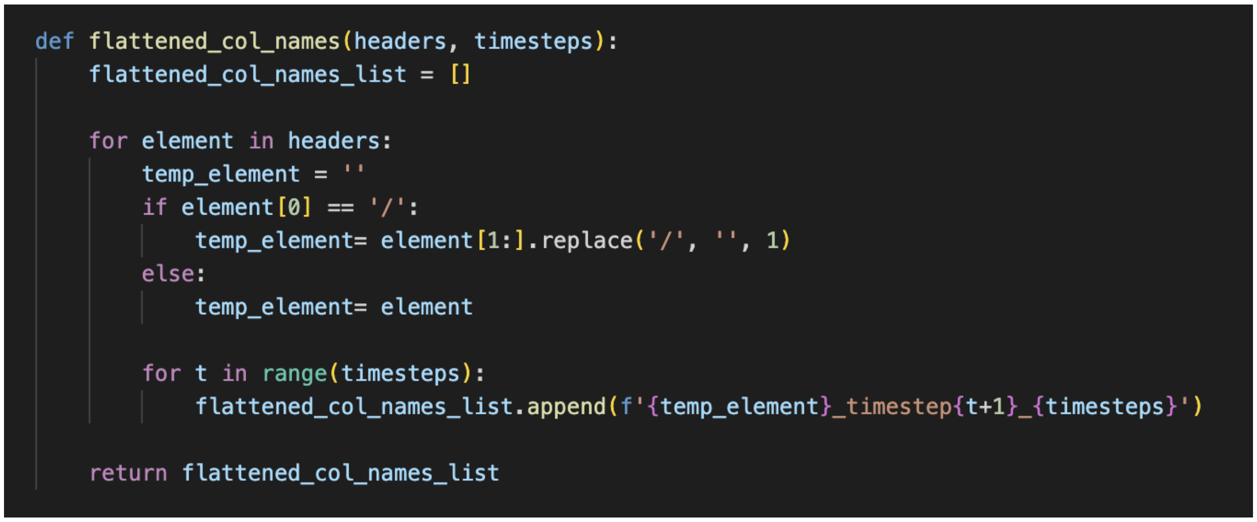
```
def prepare_dataset_seq(df, timesteps):

    df.reset_index(drop = True, inplace = True)

    df['OutageOnset'] = df['inOutage'].diff()
    outageStartIndex = df.index[df['OutageOnset'] != 0][0]

    tempTimeStepDF = df[outageStartIndex:outageStartIndex + timesteps]
    timeStepDF =  tempTimeStepDF[tempTimeStepDF.inOutage.diff().ne(0).cumsum().eq(1)]

    timeStepDF = timeStepDF.drop('OutageOnset', axis=1)
    timeStepDF = timeStepDF.drop('inOutage', axis=1)

    flattened = timeStepDF.values.flatten().tolist()

    return flattened
```

Figure 3: prepare_data_seq function

Methods (cont.)

This function takes in the DataFrame and a specified number of timesteps as parameters. The index of the row just before the outage begins is located. A new DataFrame is created containing only the rows of data that are before the outage. The `flatten()` function is run on the new DataFrame, which adjusts the DataFrame so that it becomes one-dimensional. The flattened DataFrame is returned from the `prepare_dataset_seq` function (Figure 3).

The second function, `flattened_col_names`, takes the set of columns common to all files and the timesteps, changes the list by adding every timestep to each column's name, and returns this as a list. This would be the set of Linac devices to use.



```python
def flattened_col_names(headers, timesteps):
    flattened_col_names_list = []

    for element in headers:
        temp_element = ''
        if element[0] == '/':
            temp_element= element[1:].replace('/', '', 1)
        else:
            temp_element= element

        for t in range(timesteps):
            flattened_col_names_list.append(f'{temp_element}_timestep{t+1}_{timesteps}')

    return flattened_col_names_list
```

Figure 4: flattened_col_names function

This function takes in the column headers of a DataFrame and a specified number of timestamps as parameters. From the raw outage files, some of the header names contain an extra “/” as the first character. For each of the headers where the condition of containing an “/” at the beginning of the device name applies, the “/” was removed at the beginning of the device name. Then, the headers are renamed so they include “_timestamp_” and the incremented timesteps starting from 1 all the way to the timesteps given in the parameter. The function returns a list of all the renamed columns (Figure 4).

A command line function was then created through `argparse` that takes the directory and timesteps as parameters. This will allow reusability of the code so that any directory can be used and any number of timesteps can be specified.

The final DataFrame containing the flattened data given by `prepare_dataset_seq` and the renamed list of columns given by `flattened_col_names` were combined. This DataFrame was then converted to a CSV file.

Results

In Figure 5, the table that represents the final DataFrame, 3 files were read and the timestep parameter was 10. The first column (index) of the table is the name of the current outage file that was being read. The first row (column headers) of the table contains the column names, which are the device labels from are in the format "{label}_timestep {increment}_{timesteps}", obtained from the flattened_col_names function.

	B:BLMLAM@e,52,e,0_timestep1_10	**…**	**L:BPV204@e,0A,e,0_timestep10_10**
outage_100.parquet.snappy	0.009798	…	0.009593
outage_101.parquet.snappy	0.009392	…	0.009343
outage_103.parquet.snappy	0.009392	…	0.009695

Figure 5: Table representation of example output

A condensed version of the set of headers common to all outage files is shown. This is the set of Linac devices to use for the training data in UMAP clustering:

'L:L7PSM@p,66,true', 'L:D21BPV@p,30000,true', 'L:GR3MID@e,52,e,0', 'L:SSDA3V@p,15000,true', 'L:W3MTMP@p,15000,true', 'L:V6VPAB@p,15000,true', 'L:WS2RDI@p,1000,true', 'L:K3XWF@p,15000,true',

…

'L:L0SIPH@p,1000,true', 'L:M1PRII@p,120000,true', 'L:RF4CVR@p,1000,true', 'L:W0VSET@p,66,true', 'L:V3STKI@p,15000,true', 'L:RF2RIN@p,10000,true', 'L:RF5CT4@p,1000,true', 'L:RF4RP3@p,1000,true'

Additionally, the smallest number of rows when the outage began is 215736.

Because of the `argparse` command line function, this script can be reused in the future for any directory of outage files and a specified number of timesteps.

Citations

[1] Wikipedia contributors. (2023, August 11). Artificial intelligence. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Artificial_intelligence&oldid=1169760424

[2] Wikipedia contributors. (2023b, August 11). Machine learning. Wikipedia, The Free Encyclopedia. https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=1169840378

[3] Jain, M., Amatya, V., Harrison, B., Hazelwood, K., Panapitiya, G., Pellico, W., Schupbach, B., Seiya, K., St. John, J., & Strube, J. (2022). The L-CAPE project at FNAL.