Contribution to the L-CAPE project

Jasmine Tang

Advisor: Jason St. John

Acccelerator Directorate

August 2023

Abstract

The L-CAPE project utilizes asynchronous data from thousands of Linear Accelerator devices and applies data science techniques to detect anomaly of accelerator failure before the incident, as well as automatic labels of accelerator outages. The author describes her contribution progress for the L-CAPE project this summer, as well as suggestions for future interns working on the project.

Contents

Acknowledgements

I would like to express my most sincere gratitude to Jason St. John for accepting me into the CCI program this summer and for his continued support throughout the process. His expertise and insightful advice have played a crucial role in shaping the outcome of this report, and I am truly grateful for his assistance.

I would also like to extend my thanks to Jason St. John, Carol Johnstone, Jan Strube, and Milan Jain for their invaluable guidance and mentorship, which helped me navigate the challenges of this project.

Multiple software packages/libraries contributed significantly to the project, and thus, also deserve acknowledgment:

- [VD09]-Python Programming Language and its libraries.
- [Matplotlib]-Matplotlib.
- [Har+20]-NumPy.
- [tea20]-Pandas.
- [Jai+22]- The L-CAPE project at Fermilab.

1 Introduction

The Linac provides the initial acceleration for the hadron beams used by the entire Fermilab accelerator complex. Despite the best efforts of Linac experts and accelerator operators over decades of refined and improved operation, an appreciable fraction of the intended operation remains unavailable due to unplanned downtimes. The L-CAPE project sets out to automatically label the causes of unplanned Linac outages, to allow for expert analysis and remediation. The project's further goal is to automatically forecast impending outages, using precursor signals (if any), which may enable automatic mitigation and even prevent certain outages. Finally, in cases where downtime cannot be avoided, the project will attempt to forecast downtime duration, enabling downstream power-saving measures such as RF amplitude reduction or power reduction for hundred of electromagnets and their cooling systems.

The intern, in this project report, describes her contribution progress for the L-CAPE project this summer that reflects the goal of the project, as well as suggestions for future interns working on the project.

Due to the rapid nature of the repository where the code is hosted, readers are encouraged to

2 Tools

The section talks about a variety of toolings. we used pandas with Matplotlib with Jupyter Notebook for data processing and analysis. To save data and results, we again used Jupyter Notebook with pandas and CSV files. The parquet-snappy file format is also being used for recordings of data from devices. This is first obtained from a raw HDF5 file format.

For non-numerical data (the convex/concave library) which is still columnar (for example, shapely polygon), we switched to Pickles and subsequently to sqlite3 with the Well-Known-Text format.

3 Data, Libraries and Workflow

This section details the data being used to produce this project report, as well as the workflow of the intern.

The data is stored on **gmpsai2@fnal.gov**, a shared workspace that team members can access via SSH by obtaining Kerberos tickets. Each user has their own dedicated folder where they can develop their own code.

3.1 DataGuide.csv

The DataGuide file is a csv file that encapsulates the following information.

- smooshname: The simplified name of the device, this helps generalize a device name, getting rid of their purpose and specific tag.

- Reading Device: The name of the reading device.
- Setting Device: The name of the setting device.
- RFStation: The RFStation that the device is at.
- SwitchDevice: The name of the switch device.
- Description: The metadata of the device.

The purpose of the DataGuide is that it is composed of expert matter, only containing related devices for the L-CAPE project. This helps with querying new data for training, allowing to reduce stress and noise on our model.

The csv file also comes with its own defined delimiter and extra settings via **pd.read_csv()**. In fact, often times there might not be a convention to be had about opening and saving the DataGuide.csv file that is going to be used for a PandaFrame.

This might lead to programmers and scientists using different configurations, artificially creating more friction in getting and importing data.

It is also the case that there often is a need for common queries of the data that we feel should be included in the library. For example, in Section 3.1, a programmer is concerned with the difference of the two device type. Without too much looking into the architecture of the project, the hope is that they can simply just query the difference.

Realizing the current pain point of working with the DataGuide.csv, we programmed **dataguidance.py** so that users can instantiate a class of **DataGuide** by giving the path to a dataguide.csv

3.1.1 Devices mismatch

One of the problems that occurs is that there is usually a mismatch between the reading devices and the setting devices, meaning the reading that a device produces and the reading that the operator set for that device.

In a normal condition, the difference between values of the reading device and setting device should be minimal, or near zero.

A common occurrence is sometimes when an operator set a value for a device, the model is unable to detect that the operator set that device, instead, all it sees is that there are changes in the readings of devices. Thus, it falsely reports the event as an anomaly.

We would want to extract the difference between the reading device and the setting device into data so that we can better inform the model. Thus, the need for Section 3

3.2 Workflow

I recommend setting up Remote Development with VSCode so that users can shift from developing in the terminal to developing on an IDE.

New interns when being added to the private L-CAPE repository should clone the repository to their own folder.

3.3 Workflow suggestions

The section prescribed the things I have not done but nonetheless strongly recommend new interns follow the current practices described in this subsection.

3.3.1 Environments variables

A reason to integrate environment variables is that they are modular and easy to work with. A user can rerun the same script on their machine without concerning with changing the path to a file in the Python script but can just set the environment variable to be of their needs.

This simplifies the process and ensures consistency when using scripts across different environments and users.

I recommend **dotenv** and **os.environ** for managing environments variables.

3.3.2 Deppendencies and requirements.txt

It is hard to think about dependencies regarding a large project. The team keeps a **requirements.txt** that handles dependencies for developers on the team.

For example, the Concave/Convex hull (CCCV) algorithm requires a previous version of **Shapely** that requires rolling back **NumPy** to the previous version. The intern should communicate to the team about rolling back important and needed libraries.

This stems from the fact that Python cannot handle importing the same package of different versions. A compromise is to have by asking the following questions:

- Is the library that requires rolling back previous version of dependencies important?
- How can we handle dependency version conflict? (One way is to specify a range of acceptable versions)

4 The data analyzer script

We want to figure out in a raw .hdf5 file, which devices satisfy a set of conditions and repeat this operation for thousands of devices.

With Python, this requires the utilization of the machine's cores, more specifically, with **x** amount of cores on the machine, each core repeats this same step for thousands of files:

$dg \leftarrow$ the DataGuide.csv
$colt \leftarrow$ collection
$file\_glob \leftarrow$ all data file ending in .h5
$device\_lst \leftarrow$ the list of all setting devices in dg whose device has the reading devices
for each $data\_file$ in $file\_glob$ **do**
    $keys \leftarrow$ name of control system device from $data\_file$

for each *key* in *keys* **do**
if *key* is in from *device_lst* **then**
colt[key].add(*data_file*)
end if
end for
end for
dct ← Combination of all results of each collection into a single collection.

The output is a collection represented as a dictionary, whose key A is the string representing the setting devices' name, and the value of that key is a set B of strings C. (So each value C in the set B signifies that key A was found in the file name of string C.)

After we run the script that performs the pseudo-code, the results were not very promising

Out of 654 setting devices present the DataGuide.csv of 2770 devices, 344 of those are also present

5 The Concave/Convex hull classification

The Concave/Convex hull (CCCV) classification library is developed with the idea of classifying a new 2D data point based on the hull of all the 2D data points of the same class.

The goal is to end aid operators in helping them make a decision in identifying a fault type.

The library depends on the **shapely** library in Python. It uses sqlite3 instead of Pickle to manage the polygon that shapes the hull, storing it in a **.db** file. This allows easy expansion to other languages without depending on Python-specific libraries such as Pickle.

To help realize this goal, the library supports the following function, divided by sections.

5.1 plot_hull_via_axes

Plot all the hulls based on the classes' list.

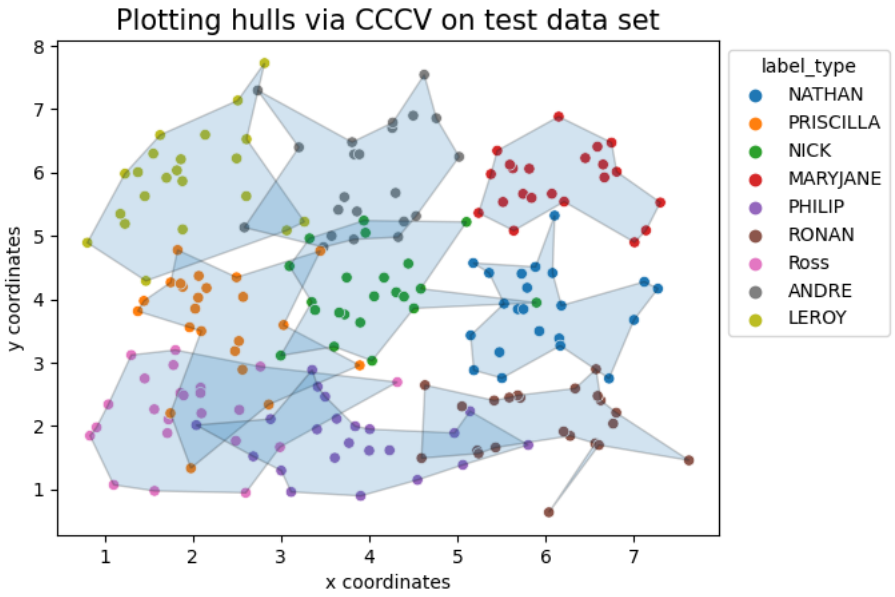

Figure 1: Demo plot showing how hulls are plotted against data points in CCCV

5.2 intersection_heat_map

Create an axes (object from matplotlib) that contains the percentage of intersection area.

For row and column:

- grid[row][column] represents the percetange of intersection area compared to row.
- grid[column][row] represents the percetange of intersection area compared to column

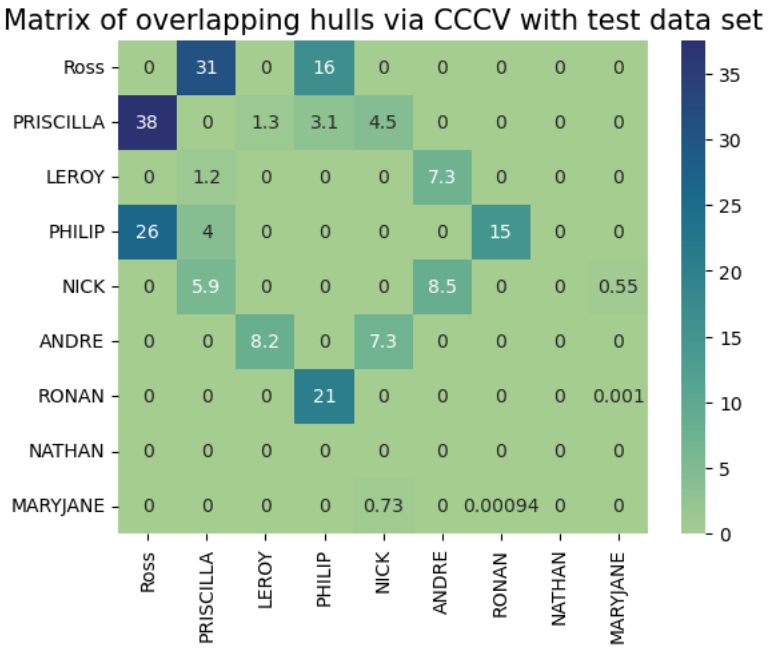

Figure 2: Demo plot showing how intersection heat map works

For example, when reading the grid, if at the row position "Ross", the column position "Priscilla" is 38%, it means that the intersection of "Ross" and "Priscilla" takes up 38% of the area of "Ross"

Users can opt to turn on interpretation = True in the function call for automatic guidance.

5.3 get_list_containing_point

get_list_containing_point gets a list of all the polygon's classes for which the point(x,y) is contained in it.

For example, if a point (4.0, 4.0) is only in 1 class's polygon named 'NICK', the list being returned will be a list ['Nick']

6 Metrics of evaluation

It is important in picking the right clustering model that we explore and define some metrics. Metrics is a tool for other developers and even the Python program to understand which models perform best.

The section discusses both labeled metrics and unlabeled metrics, as the current situation is both the shortage of labeled data as well as the absence of generating unlabeled data.

6.1 Labeled metrics

6.1.1 Homogeneity, Completeness and V-measure Score

The homogeneity score of a cluster measures the similarity between data points in each cluster. It has a range from 0 to 1, with 1 representing perfect satisfaction of homogeneity.

The completeness score of a cluster: measures if all data points of the same class are in the same cluster. [0, 1]

It is often hard to evaluate the two things at the same time, so V-measure is introduced as the harmonic mean of the two metrics.

From sklearn:

```
v = (1 + beta) * homogeneity * completeness
     / (beta * homogeneity + completeness)
```

6.1.2 Minimum count threshold

Current labeled data only has 273 data points with 93 classes. In addition, half of the classes only have 1 label for the class; this means that sometimes, these classes are often classified as noise.

We introduced the **minimum count threshold** not as a metric but to help with interpretability and noise.

6.2 Unlabeled metrics

Since the majority of data (in the future) will be unlabeled due to the lower priority of labeling data, we want a way to determine if a clustering model is good enough.

6.2.1 Silhouette Score

A measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). This score is bounded between -1 for incorrect clustering and +1 for highly dense clustering. Scores around zero indicate overlapping clusters.

To make the score compatible with other scores that have the range from 0 to 1, we try to map the distribution range of the Silhouette Score from [-1 to 1] by shifting up 1 and dividing by 2. This will help with integrating into a uniformly weighted metric.

6.2.2 S_Dbw

All the clustering metrics described have been from Scikit-learn, with little mentioning of how effective they are at measuring. In our case of massive unlabeled data, it should be said that we need even better validation of our clustering method.

A paper in 2010 validates these unlabeled (internal) metrics, including Silhouette score, Davies-Bouldin index, and S_Dbw. In regards to five different aspects of the incoming data, they note that S_Dbw performs well in all five aspects.

Unfortunately, the algorithm has not been implemented in Scikit-learn and we have not found any library that implemented the algorithm. Due to time constraint, we are unable to provide a concrete implementation of the algorithms and thus, are falling back onto the Silhouette Score.

6.3 Weighted Average

Since we have conformed all metrics into the 0 to 1 range, we can introduce a new metric that is the weighted sum of all the metrics to be included. This helps us not to be cumbersome with too many metrics.

7 Clustering: DBSCAN

We decided to try DBSCAN due to poor performance for k means clustering, decision trees and other traditional methods.

Currently, there has not been enough unlabeled data to run through DBSCAN.

8 Conclusions and future work

There is still significant work to be performed. This section presents some ideas for future work as well as the direction of the project.

A pain point of the project currently is that the model is not having the expected performance, i.e. it is still struggling with accuracy.

The L-CAPE project is going through a meeting to acquire more funding. The project will need to get more capital resources to extend past this year.

9 Miscellaneous

9.1 G4blplot

In addition to the aforementioned work, I also support other interns in their endeavors.

More specifically, I helped them get up to speed with G4Beamline, a physics simulation software, and my code base earlier in spring.

I also helped them walk through my report paper, as well as continued support for my codebase [jja], fixing any bugs and adding new features as they requested.

I also am automating documentation as well as automated tests for the project, so that my successors can keep working on this, with an infrastructure already in place for them.

9.2 Languages

I am also interested in languages and compilers. I would want to build a new G4beamline scripting language for particle physics and hope that the next summer, I would get a chance to work on some projects of similar uses.

References

[Matplotlib] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[VD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[Har+20] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[tea20] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[Jai+22] Milan Jain et al. "The L-CAPE Project at FNAL". en. In: *Proceedings of the 5th North American Particle Accelerator Conference* NAPAC2022 (2022), USA. DOI: 10.18429/JACOW-NAPAC2022-WEPA40. URL: https://jacow.org/napac2022/doi/JACoW-NAPAC2022-WEPA40.html.

[jja] jjasmine. *fermi-proj: A scripting library for g4beamline plotting/-data processing/automation*. URL: https://github.com/badumbatish/fermi_proj.