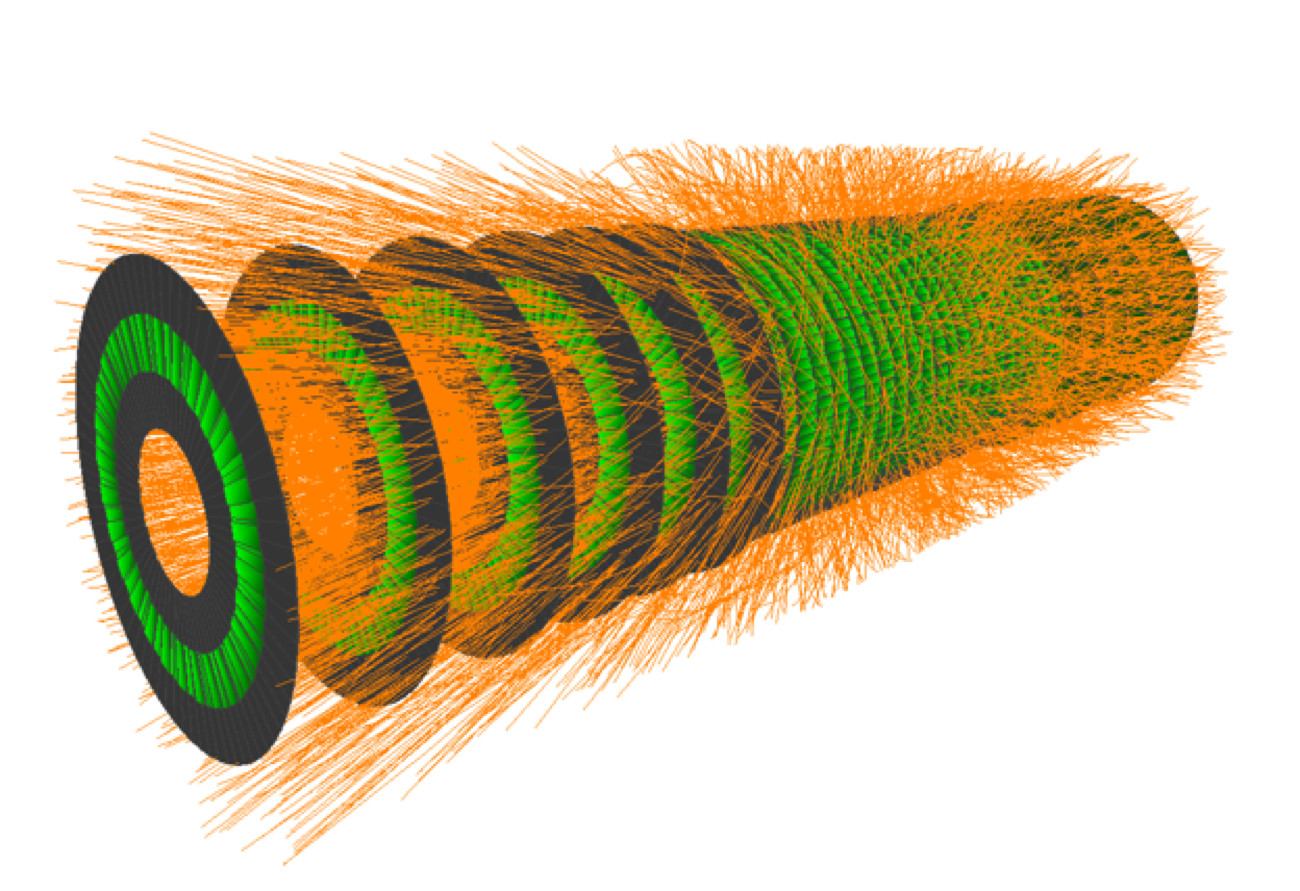
TrackML Particle Tracking Challenge



Portia Allen

Connor Caldwell

Arnaud Filliat

Johnny Murphy

October 2020

# Abstract

To discover more about particle physics at CERN, the goal of this project is to take a given set of data and develop to predict the type and trajectory of particles from various particle collisions. In these models, we explored many different model types such as SVMs, kNN and others. The main factors that determined our decisions in the modeling were the initial inputs and outputs of the models. We conducted this analysis and compared the results between the different kinds of models to choose the best model for our data. We also analyzed how the model is affected by different parameters and optimized our chosen model that way as well.

# Overview and Related Work

Subatomic particle detection has become critical to deepening our understanding of what makes up the universe. Scientists at CERN are colliding protons using the Large Hadron Collider (LHC), and observing these small scale, big-bang-esque collisions to learn more about fundamental particles. However, as observation techniques continue to increase in quality, so does the quantity of data that documents a single collision. The purpose of this project is to use machine learning algorithms to accurately categorize high energy particle collisions, and then predict new particles.

This is possible through a reconstruction of the particle events from the LHC. Using the dataset provided by CERN scientists, it is possible to break down confirmed collision events and categorize them by particle type, and then use that algorithm with unknown collision data to predict the resultant particles. Each event contains a lot of data, such as the coordinates of the collision and the particle charges, and then the trajectories of the post-collision particles. Therefore, it is critical to have a higher degree of understanding of what the data represents, in order to build accurate models and correctly interpret the training data, as well as categorize unknown data using the same parameters.

By creating this machine learning algorithm, we can more efficiently and effectively track and categorize high energy particle collisions. We can gain a better understanding of the particles, how they collide, and what particles they break into after the collision. Additionally, several members of our group have a deep background in physics, so this dataset is of particular interest. Our personal connection to the discipline can be more motivating for the completion of this project. The application of data science to this project is extremely straightforward – given some dataset, we have to parse through it, train a machine learning algorithm with some of it, test it using the rest of the data, and then refining our model to be the most accurate it can be. Through this process, we can develop a better understanding and ability to apply data science/machine learning concepts to real-world data, while also gaining a deeper appreciation for and understanding of current particle physics research.

It should be noted that this dataset was used as part of a machine learning algorithm competition in 2018. The goal was to write an algorithm with the highest accuracy, and then highest throughput (speed relative to accuracy). Several people who placed highly in the competition have posted their answers and solution process online, so that other people can understand how to approach and execute an algorithm like this. Therefore, our group can make a solid attempt at it with our more limited data science skills, and then compare to those who have more experience in the field; from this, we can see what we could improve, as well as gain more insight into applied data science.

The winning solutions used a mix of both machine learning and geometry/statistical analysis. The #1 solution used only statistics and 3D geometrical analysis techniques, while several of the other top solutions used deep neural networks, HDBSCAN, and linear regression. It should be noted that the highest scoring solutions all used machine learning algorithms to sort through the data for trimming and geometrical analysis for the actual classification, rather than machine learning for all of it.

# Data Acquisition

The dataset that we intend to work with is the Particle Tracking dataset from the Large Hadron Collider at CERN. Since this data was made publicly available by CERN for the TrackML Particle Tracking Challenge in 2018, there are no restrictions on our use or sharing of the data.

We are given a test dataset with 125 collision events upon which to evaluate our model; a training dataset, 8850 events split into five different files, as well as a sample set (the first 100 events from the training dataset); and some informational files about the geometry of the detectors. These are all provided by CERN in zipped .csv files, that can be downloaded and then read into a Pandas DataFrame for ease of access and manipulability. Additionally, CERN provides a trackml python library in GitHub to simplify some of the data handling, which will most likely be used extensively to aid in data visualization and processing.

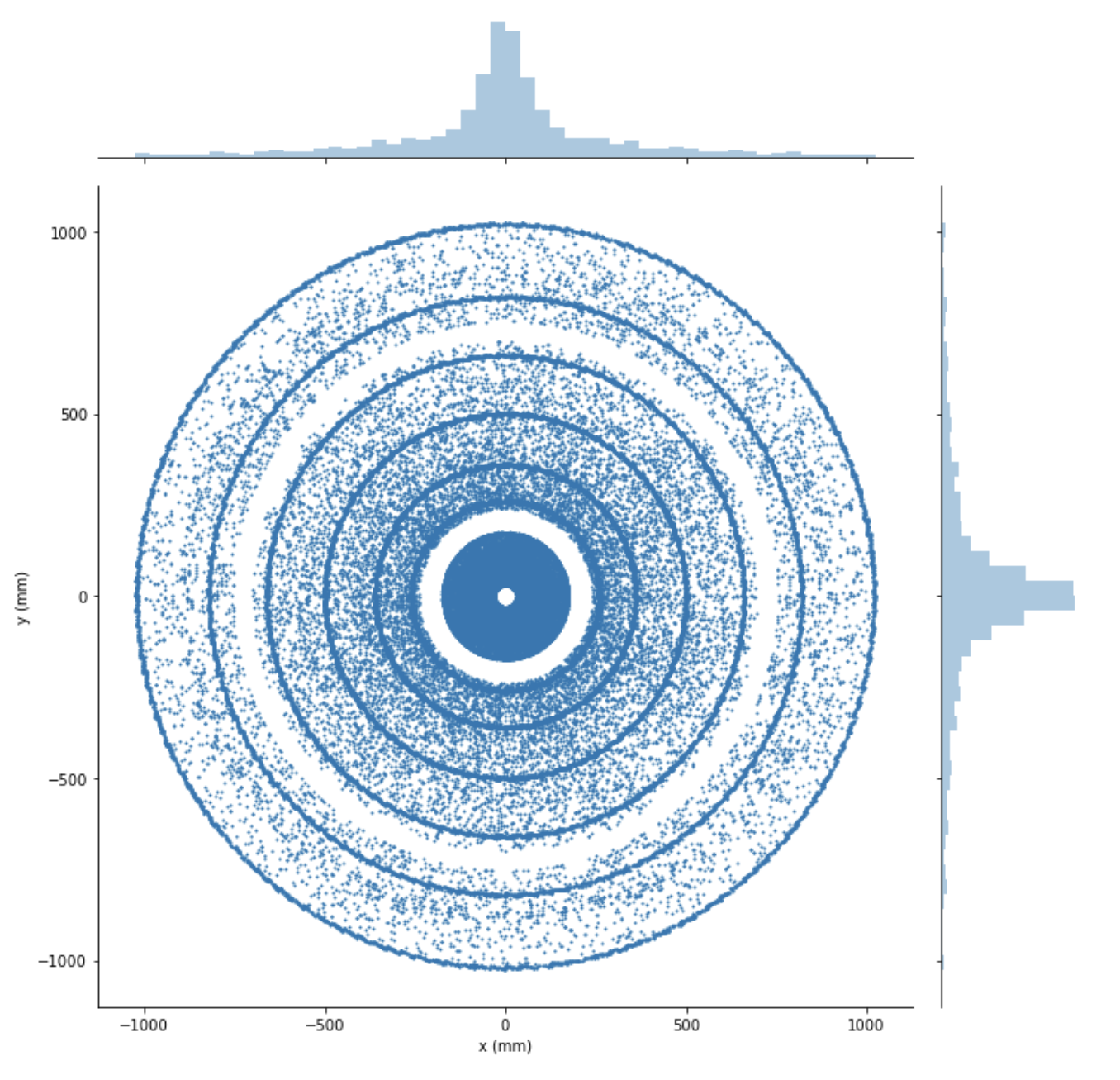
Each event has four associated files, containing the hits, the hit cells, the particles, and the ground truth for the event. The hit files contain the identification numbers for the hit itself and the detector group/layer/module location of the hit, as well as the x-y-z coordinates for the hit. The truth files have the hit identification number, the particle identification number, the true hit location and particle momentum, and the weight of the hit (for the scoring metric). The particle files contain the particle ID number, the particle type, initial position and momentum, charge of the particle, and number of hits from this particle. The hit cell files contain the hit ID number, how much charge a particle has deposited on the cell, and the cell coordinates. The cells are the smallest positional identifier on the detectors, and can be used to more accurately track association between hits and particles.

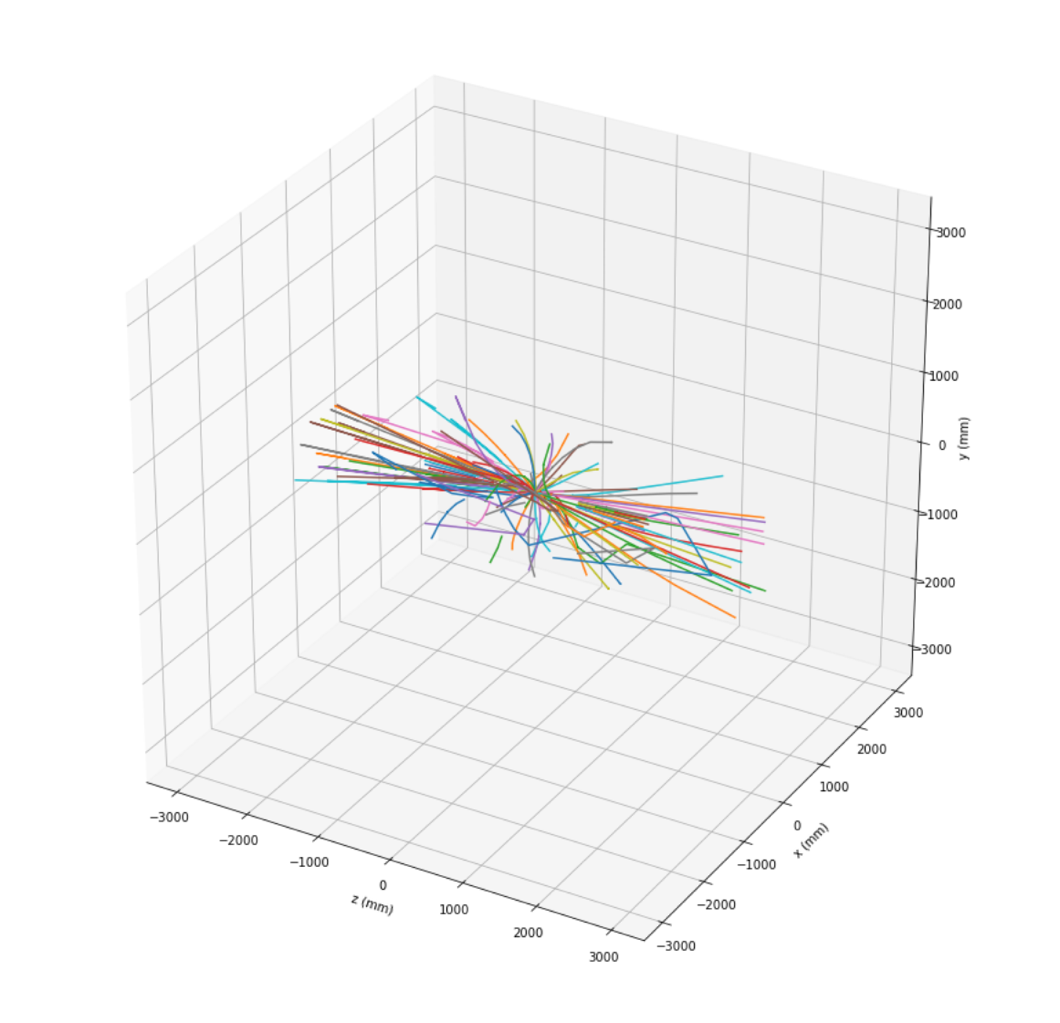
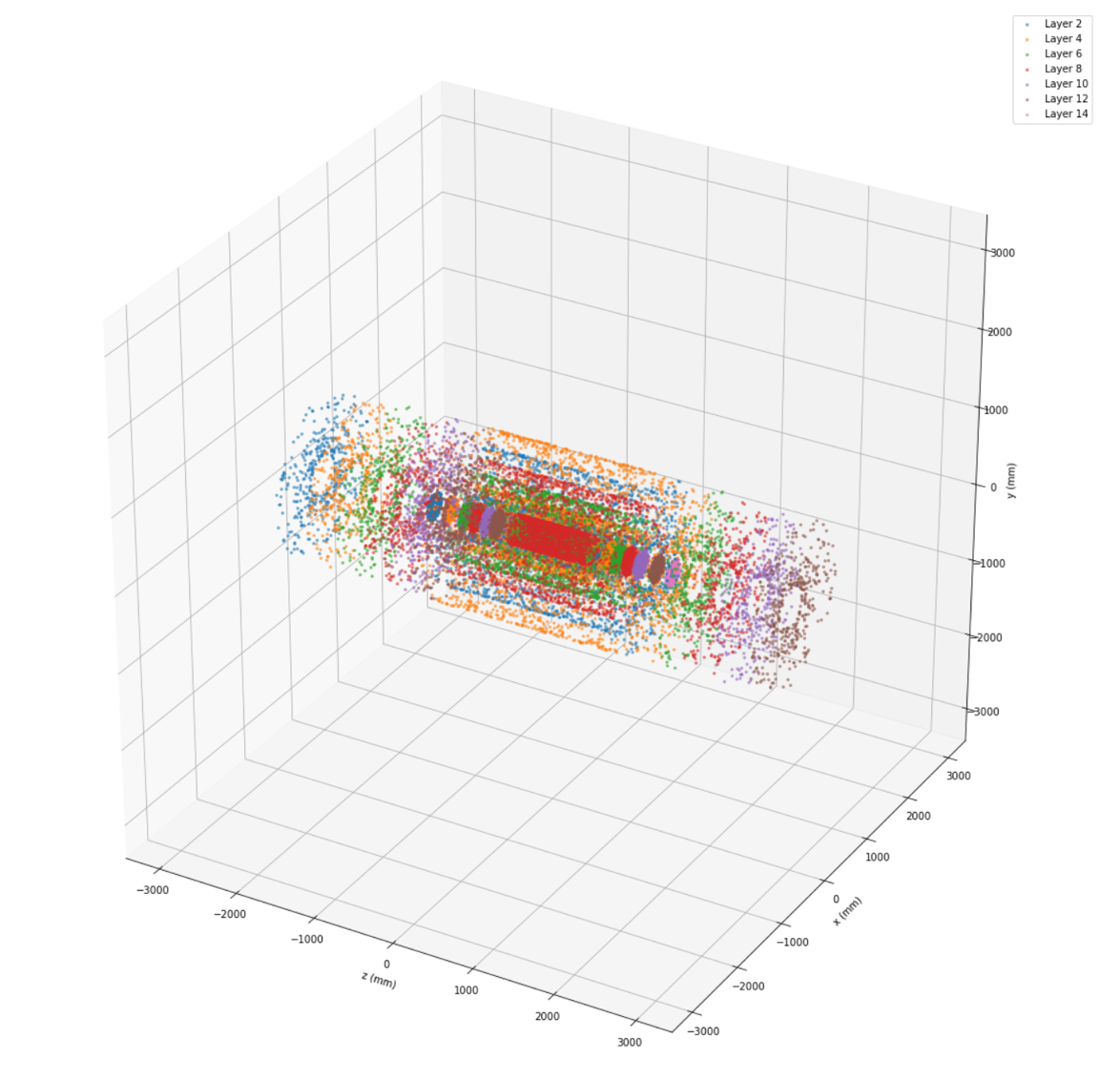
The geometry of the detectors is important information to know because the detector is built from concentric silicon slabs that have been subdivided several times. The largest groups are volumes, subdivided into layers, which are then divided into modules, which are made up of cells. Each of those have ID numbers except the cells, which have a gridded identification system. Each module has a different local position and orientation, so a transformation must be made between the local coordinates of the hit and the global coordinates of the hit to get the actual path of the particle.

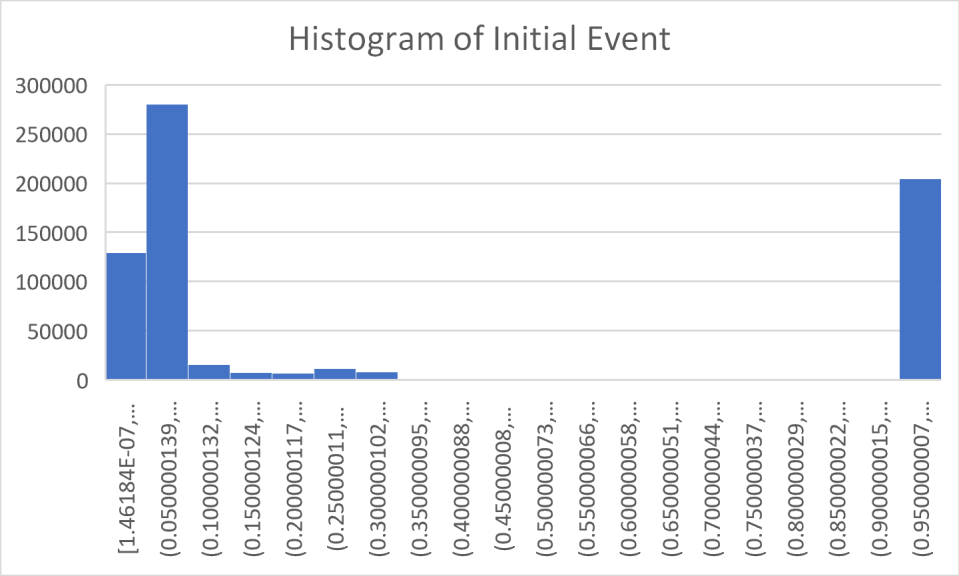
# Preprocessing

The main preprocessing goals that we took were to hopefully eliminate outliers and reduce the quantity of data bring processed to allow our computers to run it all the way through. We went about this by first normalizing the data of a single event, which still had plenty of data, and then reduced the amount of data in that event to different amounts varying on the test being done. The basic range that was used was around 5-20%, which might affect the integrity of the data, but when there are hundreds of thousands of data points, we have to reduce the amount of data and features to allow our computers to run these models. Once both of these were done, we are able to actually test and run our data sets, allowing us to find the true patterns in our models.

Below is a graph of the X values compared to the Y values in one event. As you can see this looks like a sphere which is expected since the CERN is a cylindrical model, and this allows for us to view the points in which a data point can be constrained in, and then shows that there are a couple of rings where it looks like some events where not recorded, which could help us in our training in order to solve some issues that could be a cause for concern in this model. Additionally, on each axis is a histogram of the number of hits vs where on the respective axis it hit; this clearly shows that the center detector layers had the most number of hits, which might make it easier to discard some of the data for training.



The below visualization is a 3D plot of the x, y, and z values for a single event. This clearly shows the geometry of the particle detector – a cylinder with several layers – and is an indicator of how difficult it might be to plot these tracks and match them to particles. Below the 3D scatterplot of the hits on the detector is a 3D line chart that shows particular tracks. These tracks come from the truth file, and show what we have to teach our model to predict and then match with the particle IDs.

This histogram below shows the values of this hit as a histogram with about 20 bins. This shows that the data is very skewed to the right, which means a certain particle could be much heavier, classifying it as something else, such as the Higgs Boson. The data on the left can be abstracted to particles such as beta particles, which are lighter and much more abundant, same with Z particles as well.

# Model Selection

The main algorithm we want to use is a machine learning technique using support vector machines. These will allow us to be able to predict where new data points will be using previous data points, which is perfect for tracking a 3D particle. Since the particles we are using are in a multidimensional space, the best way to use the SVM software is to utilize the kernel function, allowing for nonlinear models to be produced.

In this case the nonlinear models would be most useful in the X and Y position of the particles, allowing us to block off different groups and relate them to other features, such as the X and Y velocity. This will hopefully allow for more accurate predictions while maintaining data integrity.

SciKit-Learn has multiple functions that can be utilized for SVM machine learning, including the kernel function as mentioned prior, as well as a C variable that allows for flexibility. This prevents overfitting or underfitting allowing a unique fit for each model. The leniency in our function will have to be tested, since the particles are semi-random and are hard to get a locked down number on. It will likely be in the range of C = .1-1 area since there are so many data points.

Due to the random nature of particles and how they move another possible algorithm to use would be an unsupervised learning algorithm. We were thinking of using k-means as well to split the clustering and possibly look at reducing dimension to only the most important features. These features could include the particles’ initial velocities and their charge. This would enable a more “learned” approach which could be significant but due to the large amount of data may be impractical because of the time to run.

# 5 Results and Evaluation

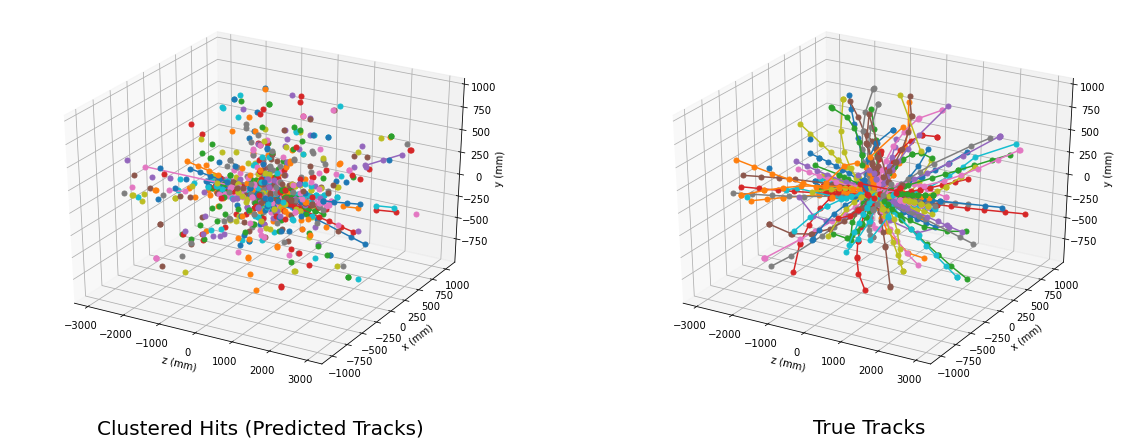
Originally we were going to use an SVC model to train and then predict the particle tracks. However, the time complexity of the SVC machine learning module is O(n2), meaning that the sample size, n, will take a heavy toll on the computing system with a large amount of data. In our case the data reached well over 845,000 units in just the hits folder, in just one part of the large data set. Since we don't have industrial computers or days to wait for a model to train, we decided to normalize the data given and reduce the total amount run into training the model. This overall reduced performance dramatically, but it was the best we could do due to the circumstances. We ended up reducing the data to about 1-5% of the true amount of a single event, resulting in 967-4835 cells in 7 different features. We felt that this was a reasonable amount of data since our computers could actually process this data at a relatively fast rate. From there we tested all of our different models, but each one struggled to prove to be a good estimator due to the lack of data ran through the code. In the future it would be more realistic to take on this dataset with a designated computing system instead of the materials at hand now.

There were two main scoring functions used in the analyzation of the models created. The main one used to analyze the actual data predicted using the model is the traditional scoring function that we used in class previously : using the .score function with the predicted y values using the test data set from the split earlier on. There was, however, another function that was used from an imported library (track-ml) that told us the amount of true leniency we have with the data. This is the "score\_event" function, that takes in the entirety of the event and basically gives the user a lenience that they can work with due to noise. In our case it usually sat around a 95% mark, giving us 5% lenience. This was calculated by shuffling all of the "hits" inside of our event, then reassigning the hits with some different targets within the event. This then was scored and given back to us was our lenience that could be expected. This also acted as a random test submission that could be analyzed for the integrity of the data, specifically the "truth" or the target of our model.

In this first model, we used a clusterer (DBSCAN) to develop our model. DBSCAN is short for “density-based spatial clustering of applications with noise.” It attempts to group together points that are closely packed together similarly to nearest neighbors. It is one of the most common clustering algorithms used.

By using DBSCAN, it allows for us to find the predicted tracks of the particles that we were wanting to discover and then allows for us to develop a model based around that. The mean score for these events were approximately .197, which is low for this model. A score being close to 1 would produce more accurate results, but since this model contains so much data, it seems as though this model is struggles due to the nature of the paths being quite random and based on the features input. We also tried DBSCAN, after scaling and normalizing the data based on our coordinates. This has a score with approximately 0.1481. This score isn’t too bad, but sadly not an improvement. It’s also interesting that after normalization and scaling, the model score decreased. This could be due to altering some important data that is critical to classifying the particles, but because scaling and normalization shouldn’t directly affect the content of the data, there could be a different explanation.

In addition to using DBSCAN, we also decided to use HDBSCAN, which is a clustering algorithm that extends DBSCAN by converting it into a hierarchical clustering algorithm, and then uses a technique to extract a flat clustering based on the stability of clusters. Using HDBSCAN we were able to get a score of 0.246. We also arrived at a number of clusters that wasn't insanely high nor too low (clusters = 8972) with only 19% of the samples rejected. Although not mentioned in the course, several of the highly ranked competitors in the TrackML challenge used DBSCAN and/or HDBSCAN for either data preprocessing or as their machine learning algorithm. Although our implementation of HDBSCAN was very rudimentary, it did perform the best out of the algorithms that we chose to try tackling this difficult dataset.

DBSCAN Plot (Polar Coordinates)

Chart, scatter chart

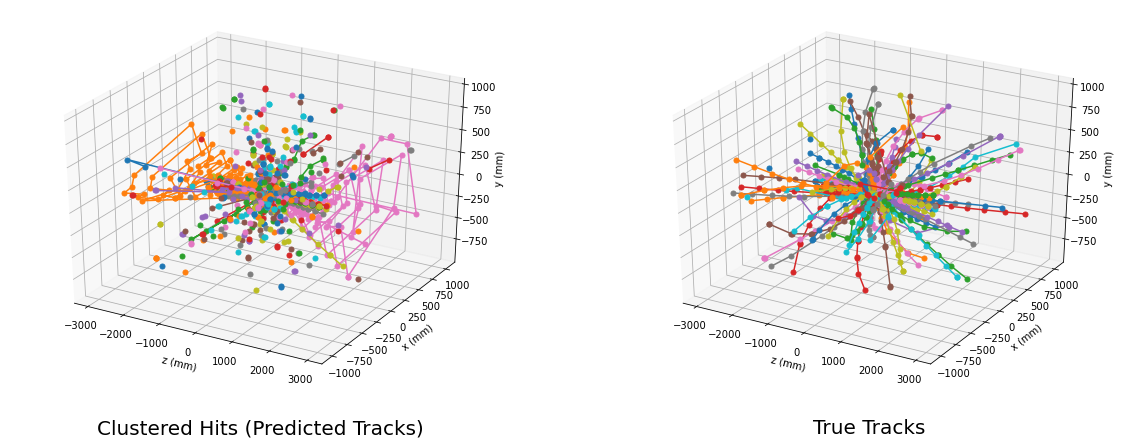
Description automatically generatedDBSCAN Plot (Polar, Scaled, Normalized)

Chart

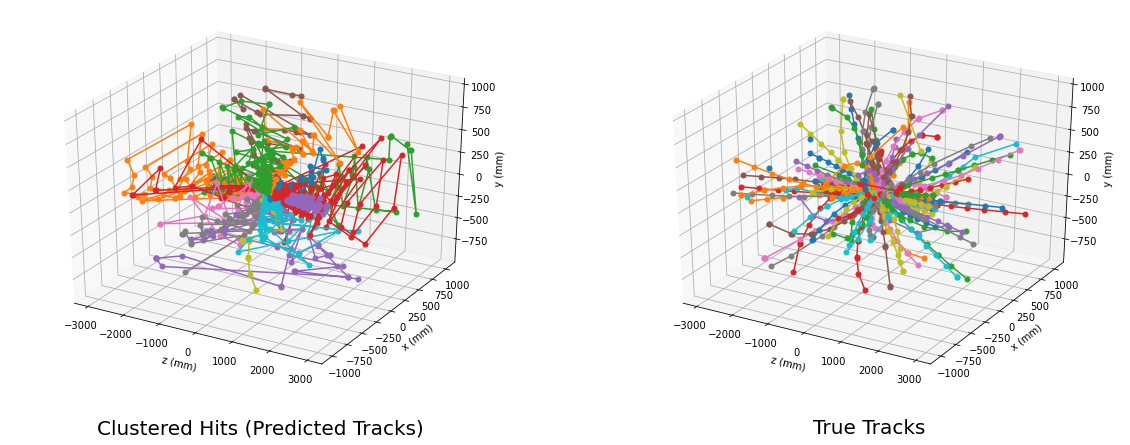
Description automatically generatedPlot for hdbscan (Polar)

We also decided to ues a second model of DBSACN, using a different number of clusters within the data by changing the minimum sampling. This allows for more samples to be clustered and creates a model such that there are more tracks. Our score for this model was also approximately 0.195, which is lower than the first model, but it does have many more clusters, this could be an example of overfitting because there were over 6,000 clusters created by this one operation.

In this next model, we changed the EPS value, meaning there is a less amount of density required to form a cluster, and then also allows for less tracks. As shown, this model does have less tracks and less clusters, but the score is greatly damaged by this, being a 0.0644. This strikingly low number can be attributed to the change in the EPS value, creating a major difference in our model.

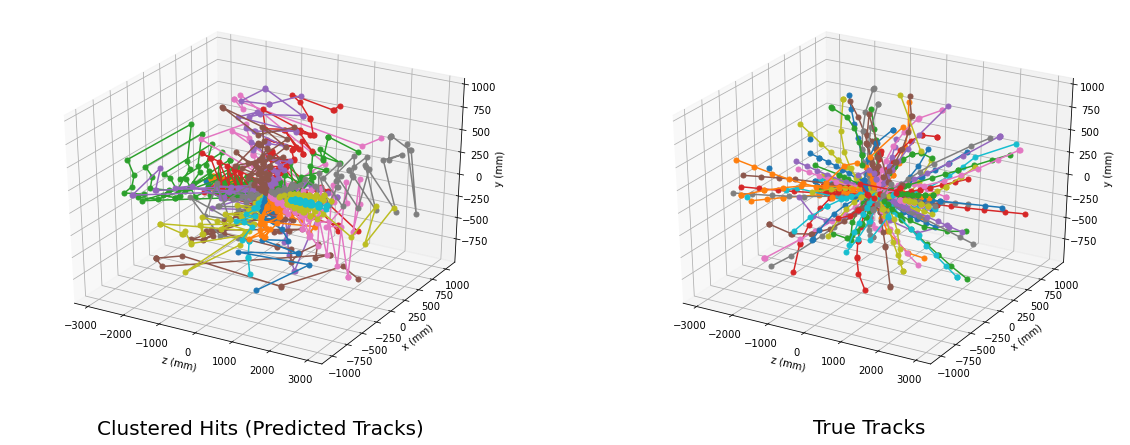
Altered EPS Plot (polar)

We also tried several other algorithms to see if supervised or unsupervised learning would be better. We originally thought that supervised learning would be the better choice, because we could train our model with labeled data and “teach” it the correct particle classifications. However, since our supervised learning algorithms were not performing well at all, we thought it might be good to at least try an unsupervised classification algorithm. Our first attempt at unsupervised learning, using the k-means algorithm, didn’t perform well at all, having a score of 0.0. From the graphic below, it is clear that k-means didn’t accurately plot the tracks, and couldn’t correctly plot the predicted tracks. We tried many different numbers for k clusters, and still couldn’t get a score above zero, as well as when using Birch, another unsupervised clustering algorithm.

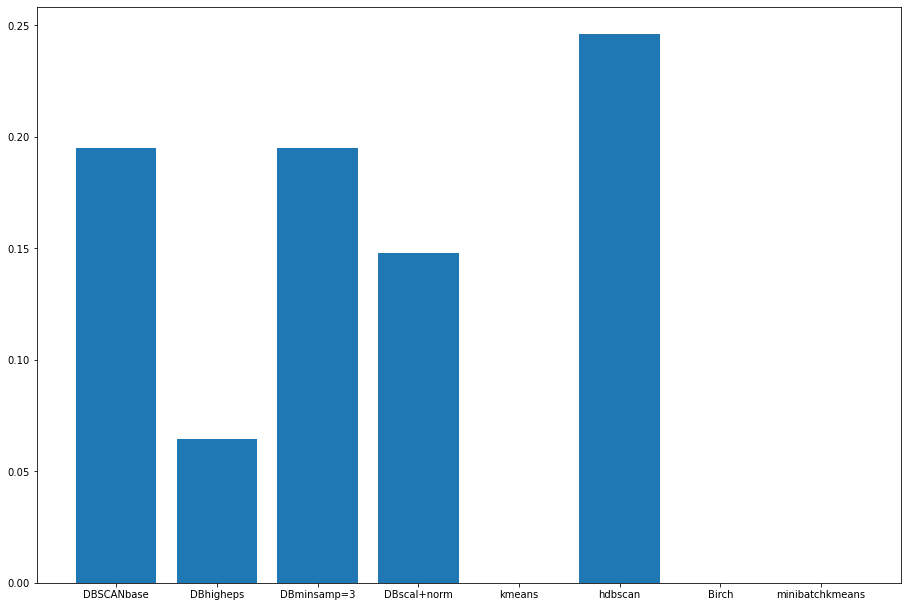
Plot for kmeans (Polar)

Chart

Description automatically generatedPlot for birch (Polar)

Plot for minibatchkmeans (Polar)

Bar graph showing various algorithm scores



Above is a histogram that compares the different scores for the various model types. Our best model was the HDBSCAN. The algorithms with a score of zero are those for which we could not get our model to score higher than zero, despite changing the training data and many of the parameters for the algorithm.

Our best score was about 25% accuracy, which overall is very poor. However, this dataset was extremely complicated to sort through, given the geometry of the detector and the data given for each particle hit. Looking back, we could have done a few things differently, like preprocessed our data slightly more effectively (through PCA or something similar), or put more time into researching HDBSCAN/DBSCAN and optimizing those parameters. Although we had more time to work on it compared to those in the competition (4 vs 1 week), our group is very new to both writing code, processing and visualizing data, and working with machine learning algorithms, unlike those who entered the competition. Despite an overall poor score, it was fascinating to see a real life application of data science techniques and how machine learning can better advance science and technology.

# Works Cited

Bonatt, J. “TrackML EDA, Etc.” *Kaggle*, Kaggle, 3 May 2018, www.kaggle.com/jbonatt/trackml-eda-etc.

Brownlee, Jason. “10 Clustering Algorithms With Python.” *Machine Learning Mastery*, 20 Aug. 2020, machinelearningmastery.com/clustering-algorithms-with-python/.

Byfone. “DBSCAN for CERN.” *Kaggle*, Kaggle, 8 May 2018, www.kaggle.com/byfone/dbscan-for-cern.

Elshamy, Wesam. “TrackML Problem Explanation and Data Exploration.” *Kaggle*, Kaggle, 9 June 2018, www.kaggle.com/wesamelshamy/trackml-problem-explanation-and-data-exploration.

Hushchyn, Mikhail. “DBSCAN Benchmark.” *Kaggle*, Kaggle, 4 May 2018, www.kaggle.com/mikhailhushchyn/dbscan-benchmark.

Hushchyn, Mikhail. “Hough Transform.” *Kaggle*, Kaggle, 4 May 2018, www.kaggle.com/mikhailhushchyn/hough-transform.

Hushchyn, Mikhail. “KNN Approach.” *Kaggle*, Kaggle, 4 May 2018, www.kaggle.com/mikhailhushchyn/knn-approach.

Lathwal. “A Very Extensive EDA of Physics Particles : Plotly.” *Kaggle*, Kaggle, 4 May 2018, www.kaggle.com/codename007/a-very-extensive-eda-of-physics-particles-plotly.

Pandya, Pranav. “A Beginner's Guide to CERN's TrackML Challenge.” *Kaggle*, Kaggle, 9 May 2018, www.kaggle.com/pranav84/a-beginner-s-guide-to-cern-s-trackml-challenge.

Zinovev, Alexander. “HDBSCAN and Scaling of the Coordinates.” *Kaggle*, Kaggle, 18 May 2018, www.kaggle.com/afaist/hdbscan-and-scaling-of-the-coordinates.