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.

In this first model, we used a clusterer (DBSCAN) to develop our model. By doing this, 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 .199, 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 predicting the wrong values, based on the features input.

Chart, scatter chart

Description automatically generated

There were two main scoring functions used in the analyzation of the models created using SVM. The main one used to analyze the actual data predicted using the model is the traditional scoring function that we have used as a 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 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.

We then decided to us a second model of a different number of clusters within the data, this then uses a different number of clusters within the model. 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 .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.

Chart

Description automatically generated

Chart

Description automatically generatedIn 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.

The model below shows with different model with scaling and transforming the model based on our coordinates. This then allows for something similar and then creates a score with approximately .1481. This score isn’t too bad, but it seems as the EPS really effects on how the score is produced.

Chart, scatter chart

Description automatically generated

We also tried several other algorithms to see if supervised or unsupervised learning would be better. 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.

Chart

Description automatically generated

Chart, bar chart

Description automatically generated

Above is a histogram that compares the different scores for the various model types. Our best model was the HDBSCAN, which is ARNAUD INSERT EXPLANATION HERE. WHAT IS IT AND WHY DO WE THINK IT DID BEST.

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. 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.