Machine Learning

Final Project: TrackML

Barcelona, June 20th 2020

Jake Watson

Contents

1 Introduction

At CERN, scientists are attempting to probe the building blocks of our world. By colliding protons together at close to the speed of light, and sifting through the wreckage, we can discover new physics.

These collisions generate titanic quantities of data, at a rate of hundreds of millions of events per second. To comb through this data, new and faster algorithms are needed. To address this, CERN started a competition in partnership with Kaggle [1], to attract new and innovative algorithm designs. The challenge was to reconstruct particle tracks from detector data, generated by a realistic simulation of the products of a series of collisions. In this project, I detail one approach to solving this problem, using an unsupervised clustering algorithm.

2 Track Reconstruction

At the LHC, proton beams are circulated around a 26.7 km tunnel, which accelerates them to 0.999999999c, almost the speed of light. Two beams travelling in opposite directions are then collided inside sections of the tunnel equipped with detectors, causing a shower of subatomic particles to be produced. These particles travel through the detectors and interact with them. These interactions are recorded as *hits*.

By grouping the hits by their particle, we can reconstruct the track of the particle through the detector. This path can be analysed to reveal information about the particle, such as its starting energy, its charge, and other properties, from which we can draw conclusions about the physical process that produced it. This is the main idea behind the use of particle colliders.

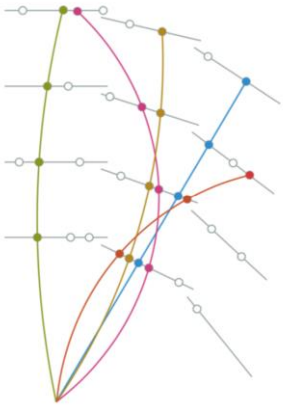

Figure 1: Connecting the dots: the core concept of track reconstruction.

However, grouping the hits correctly is a difficult problem. We do not know which particle caused the hit. As you can see in Fig.3, there are a very large number of particles travelling through the detector, so separating the hits correctly can be very difficult and computationally expensive

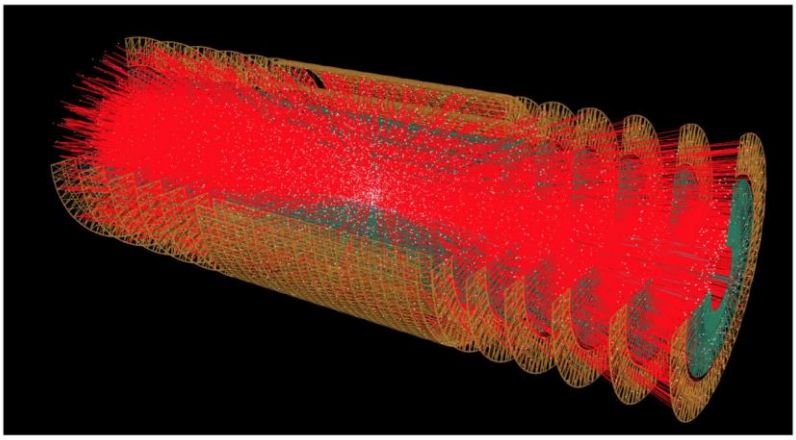

Figure 2: Trajectories of particles through the detector.

2.1 Dataset

The TrackML [1] dataset consists of 5 train files, and one test file. The train files contain 8850 events, split evenly. The combined size of this dataset is 75.8 GB. Each event corresponds to one beam collision, or approximately 200 proton-proton collisions. The information for each event is separated into multiple csv files, as follows:

```
event000000000-hits.csv
event000000000-cells.csv
event000000000-particles.csv
event000000000-truth.csv
```

2.1.1 Hits

The hit file contains the position of each hit, both cartesian coordinates and the part of the detector that the hit occurred in.

- hit_id: integer identifier for the hit. Numbering restarts for each event.
- x, y, z: cartesian coordinates for each hit.
- volume: integer identifier for the detector group.
- layer_id: integer identifier for the layer in the detector group.
- module_id: integer identifier for the module in the layer.

2.1.2 Cells

The cells are the smallest units of the detector, equivalent to pixels in a screen. This file provides information about the interaction of the hit with the detector.

- hit_id: integer identifier for the hit. Numbering restarts for each event.
- ch0, ch1: channel identifiers for each hit.
- value: amount of energy deposited by the hit in the module.

2.1.3 Particles

The particles file contains information about each particle that travels through the detector. Each particle corresponds to one track.

- particle_id: integer identifier for the particle. Numbering restarts for each event.
- vx, vy, vz: initial position of the particle in the detector, in cartesian coordinates (in millimetres).
- px, py, pz: initial momentum of the particle along each axis (in GeV/c).
- q: charge of the particle (in multiples of the electron charge).
- nhits: number of hits caused by this particle.

2.1.4 Truth

The truth file contains the correct grouping of each hit into tracks. It links these tracks to the particles that caused these tracks.

- hit_id: integer identifier for the hit. Numbering restarts for each event.
- particle_id: the identifier of the particle that caused the hit. A value of zero means that the hit was caused by random noise in the detector.
- tx, ty, tz: true position of the interaction between the particle and the detector.
- tpx, tpy, tpz: momentum of the particle along each axis, at the position of interaction (in GeV/c).
- weight: weight of this hit in the scoring metric. The sum of weights in an event is 1.

2.1.5 Detectors

This file contains physical information about the parts of the detector. For each part, it contains the dimensions and rotations from the origin for each module, layer, and volume in the detector.

3 Data Exploration

3.1 Data Loading

To start with, the four files corresponding to the event, plus the file detailing the detector layout, were loaded into R. As you can see, the number of hits and ground truth tracks are very large, even for a single event.

```
> dim(hits)
[1] 120939      7
> dim(cells)
[1] 664996      4
> dim(particles)
[1] 12263     9
> dim(truth)
[1] 120939      9
```

3.2 Hits

In Figure. 4 we can see that the x and y distributions are tightly clustered around 0, with approximately a normal distribution. The z distribution, along the axis of the detector, is significantly more spread out. We can also plot the distribution of hits within the 3D space.

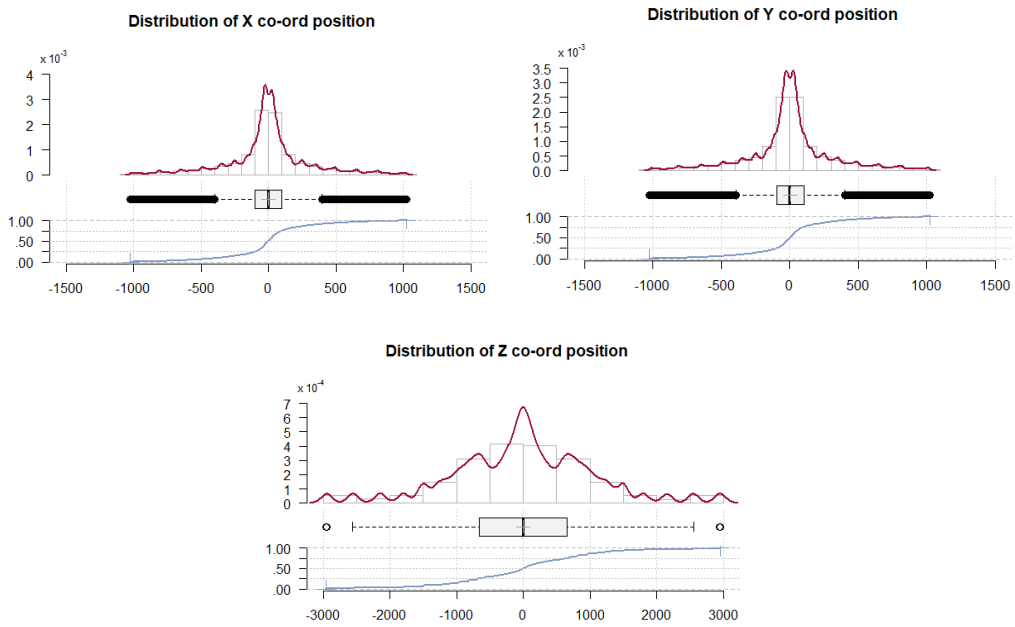

Figure 3: Distributions of cartesian coordinates of hits.

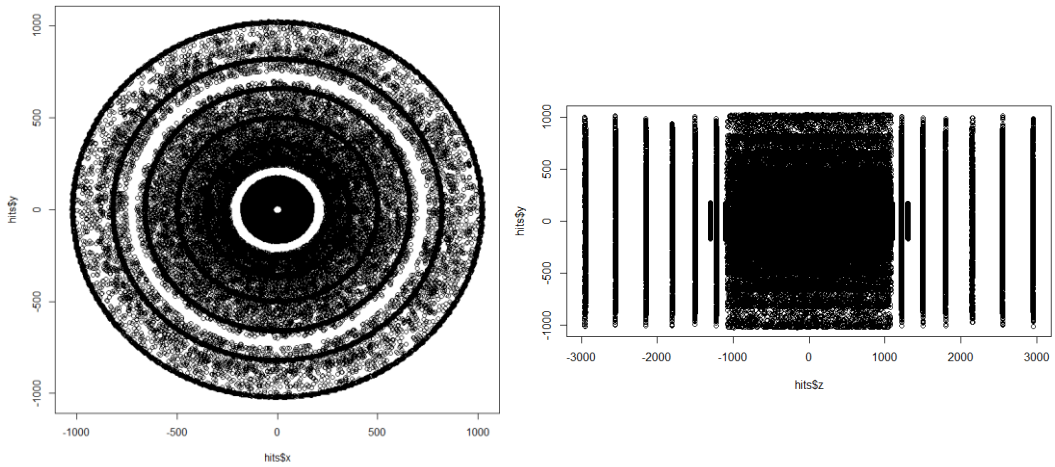

Figure 4: x-y and z-y distributions of hits.

From the above plots, we can clearly see the structure of the detector. It is the shape of a long cylinder, with layers of detection modules arranged in concentric rings around the centre. The white dot in the centre of the x-y plane is the location of the beam pipe, where the protons are collided. The particles produced by the collisions then travel out in all directions. The majority are detected near the centre of the detector, where the most detection modules are located. In the z-y plane, we can see more of the structure of the detector. The centre of the z-axis has the largest concentration of detection modules, while each end has far fewer, separated widely. This is because the collisions occur close to the origin, so most particles will travel through a space close to the origin in the z-axis.

3.3 Particles

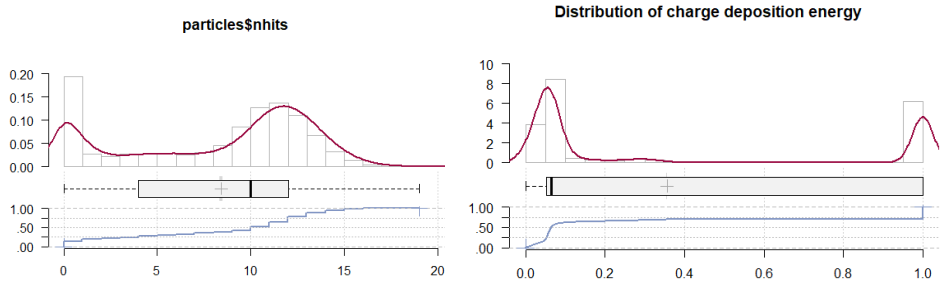

Figure 5: Distribution of nhits and energy deposited for particles.

We see that there are two peaks for the nhits value per particle, at 0 and 12, and that there are no values greater than 20. This value is the length of a track for a particle. Many particles go right through the detector and leave no trace. We also see two clear peaks for the energy deposited. Again, there is a peak close to zero, so most particles deposit very little energy, while some deposit a large amount. There is little middle ground.

4 Clustering

The problem of track reconstruction can be treated as a clustering problem. We have a large number of hits that must be grouped into tracks, with no duplicates in other tracks. The hits in their raw state are not suitable – they must be transformed.

4.1 Helical Tracks

The particles in the detector move under the influence of a strong magnetic field. This causes their paths to form helixes along the z-axis.

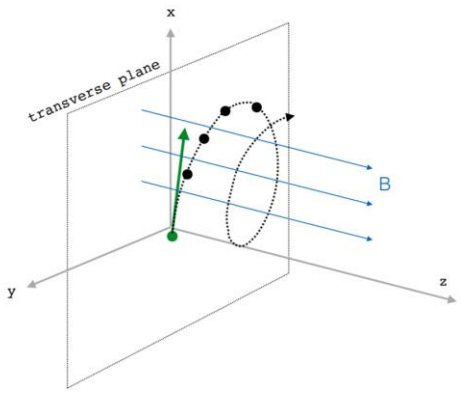

Figure 6: Illustration of helical track, B represents magnetic field.

An ideal helix can be described by its radius, its tangential angle in the x-y plane, and by its slope (the rate of change in the z-axis divided by the rate of change in x-y plane). These values are constant for an ideal helical track, and so would present ideal features for separating hits. If several hits fell on the same helix, it is safe to assume they were caused by the same particle, and so an be safely clustered.

These values cannot be directly calculated from a single hit, as they depend on the particles initial momentum and direction of travel. However, we can attempt to fit various potential helices to particles, and so discover likely values of the helix parameters for the track that connects these hits [2]. To calculate the tangential angle ϑ of the helix for a hit, we first represent the hit in polar coordinates.

$$\varphi = \arctan\left(\frac{y}{x}\right)$$

$$r = \sqrt{x^2 + y^2}$$

$$\vartheta = \varphi + \Delta\theta = \varphi + \arcsin\left(\frac{r}{2R}\right), \qquad R = radius\ of\ helix$$

However, we do not know the initial radius of the helix. Therefore, for a set of hits, we must scan over potential helix radii to discover the helix that fits the largest number of hits. This gives an approximation of the tangential angle, where i is an integer varied over many values.

$$\Delta\theta = \frac{r + 0.000005r^2}{500i}$$

4.1.1 Features

By transforming the x, y, z coordinates of hits into polar coordinates, and by computing helix-invariant metrics, we can obtain useful clustering metrics for each hit. These metrics are related to the Hough transform.

- $\sin(\vartheta), \cos(\vartheta)$: represents the tangential angle of the helix. We take the sine and cosine to account for periodicity of the angle, and so avoid numerical instability
- $\frac{x}{r}, \frac{y}{r}, \frac{z}{r}$: invariant for high-momentum particles

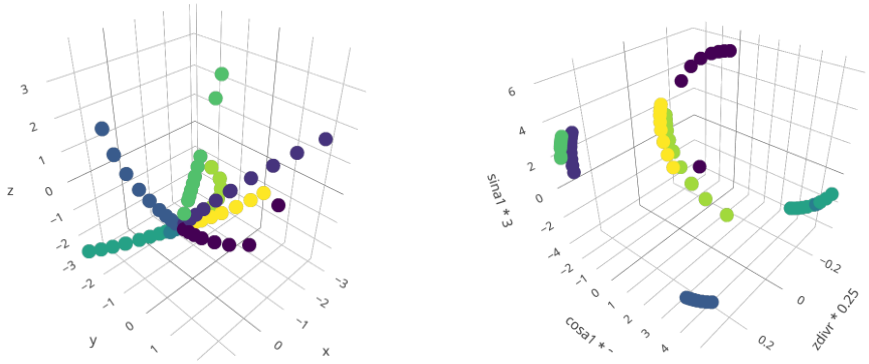

Figure 7: Left: cartesian representation of tracks. Right: helix-invariant representation of tracks Note the separation of each track in the invariant representation.

4.2 Clustering

```
1. Compute helix-invariant features for each hit.
2. Loop for n_iters:
      a. Calculate new values of tangential angle features.
      b. Scale the features.
      c. Perform a clustering of hits using the DBSCAN
         algorithm.
      d. Count size of each new cluster as N2.
      e. For each hit: if the new cluster size is greater than
         the old cluster size N1, and less than 20, accept
         the new cluster assignment.
3. Return clustered hits.
```

To compute the clusters for each event, I followed the above algorithm. The clustering prioritises larger clusters, but performs no evaluation of the quality of a cluster after it is formed.

4.2.1 DBSCAN

DBSCAN (Density Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm. This algorithm works as follows:

- Find the points within a radius ε of every point. Label those points with more than *minPts* as core points.
- Find the connected points of core points, ignoring non-core points.
- For each non-core point, assign to a cluster if it is within ε. Else assign to noise.

The advantage of using this algorithm is that it can find clusters of any shape, and it does not require the number of clusters to be set in advance (as this number is unknown in our case). Our clusters may take the shape of lines, so a density-based approach seems like the way to go. Finally, it is fast – an important consideration given the size of our dataset.

4.3 Scoring Metric

The scoring metric for the competition was used in this project [1]. This score is defined as the intersection between the reconstructed tracks and the ground truth particles, normalized to one per event. Only those clusters with more than 50% of its hits correctly matched to a single particle have score above zero. Additionally, the cluster must contain more than 50% of the matching particle's hits. This is known as the double majority rule.

The score of the track that matches this rule is then the sum of the weights of the points the intersection between the track and the particle. This per-track score is then summed over the event, normalized to one. An R implementation (the original function was in Python) is below: [3]

```
score <- function(labelled, truth) {
  tmp = merge(labelled, truth[,.(hit_id, particle_id, weight)])
  tmp[, Np:=.N, by=particle_id]
  tmp[, Nt:=.N, by=track_id]
  tmp[, Ntp:=.N, by=list(track_id, particle_id)]
  tmp[, r1:=Ntp/Nt]
  tmp[, r2:=Ntp/Np]
  sum(tmp[r1>0.5 & r2 > 0.5, weight])
```

5 Results

5.1 Maximum Score on Training Event

To develop the clustering algorithm, the score was evaluated for a single event, the same one used as visualisation, `event000001000`. A maximum score of 0.5279 was obtained using the clustering detailed above, for a single event.

5.1.1 Cluster Sizes

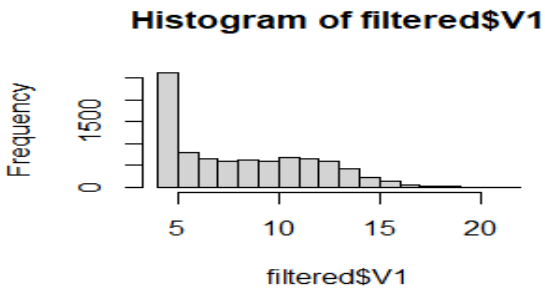

Figure 8: Found cluster sizes.

By filtering the clusters to remove the large number of smaller, spurious clusters, we can see that there is a small peak at low cluster sizes, and at around 12 hits per cluster, with a slow drop to 20. This is approximately the same behaviour as the ground truth cluster sizes. However, our clustering algorithm clearly clusters together too many small tracks.

5.1.2 Parameter Optimisation

The clustering algorithm takes 8 parameters. To optimise these, Bayesian optimisation was performed, using the `rBayesianOptimization` package.

- w1-w6: weights for each helix-invariant feature.
- n_iters: value of n_iter in the clustering algorithm.
- ep: radius of points for clustering in DBSCAN.

```
OPT <- BayesianOptimization(single4opt,
  bounds = list(w1 = c(1.0, 3.0), w2 = c(1.0, 3.0), w3 = c(0.5, 1.0),
                w4 = c(0.1, 0.5), w5 = c(0.01, 0.05), w6 = c(0.01, 0.05),
                n_iters = c(150L, 200L), ep = c(0.001, 0.01)),

        init_points = 3, n_iter = 20, acq = "ucb", kappa = 2.576,
        eps = 0.0,verbose = TRUE)
```

```
Best Parameters Found:
Round = 15
w1 = 2.7321 w2 = 3.0000 w3 = 0.9561 w4 = 0.5000 w5 = 0.0218 w6 = 0.0289
n_iters = 200.0000  ep = 0.0090
Value = 0.5279
```

5.2 Maximum Score on Test Events

To evaluate the algorithm's performance on unseen events, it was evaluated for ten more, unseen, events. To speed this up, a parallel version was implemented. It was not tested on the provided Test files, as these lack ground truth values. These files were provided so that Kaggle could compute them privately, maintaining fairness in the competition, but are useless for our purposes.

```
scores <- foreach (event=0:9, .combine=rbind) %dopar% {
  n_event = 1001 + event
  path = paste0("../train_1/event00000", n_event)
  truth = data.table(read.csv(paste0(path, "-truth.csv"), header=TRUE, sep=","))

  clustered <- cluster_transform3(path,
                                  w1=3.0, w2=3.0, w3=1.0, w4=0.2521, w5=0.0211, w6=0.0474,
                                  n_iters = 190,
                                  ep = 0.0088)

  labelled <- clustered[,.(n_event, hit_id, track_id)]
  s = score(labelled, truth)
  return(s)
}
```

Event	Score
1001	0.5378
1002	0.5089
1003	0.5295
1004	0.5228
1005	0.5294
1006	0.5276
1007	0.5267
1008	0.5162
1009	0.5327
1010	0.5162
Average	0.5245

The differences between these test events and the training event are negligible. Each event contains a very large amount of data, approximately 100,000 rows. The composition of clusters is uniformly distributed for each event, so we see very little difference between events, or by increasing the number of events.

5.2.1 Comparison to Kaggle Leaderboard

This score places this approach at position 126/651.The majority of the highest scoring approaches used some variant of supervised learning, or did heavy extra postprocessing of clusters, in order to determine which tracks exhibited physically unreasonable behaviour.

5.3 Failed Approaches

5.3.1 Helix parameter sampling

Instead of iterating over many values of the helix tangential angle, it was attempted to directly sample this value from the training set ground truth, as well as other parameters. This gave rise to numerical instability in the features so could not be continued.

```
helix_params =  data.frame()
for (ii in 10:99){
  print(ii)
  truth = data.table(read.csv(paste0(path, "event0000010", ii, "-truth.csv"), header=TRUE,
sep=","))
  hits = data.table(read.csv(paste0(path, "event0000010", ii, "-hits.csv"), header=TRUE,
sep=","))
  cells = data.table(read.csv(paste0(path, "event0000010", ii, "-cells.csv"), header=TRUE,
sep=","))
  particles = data.table(read.csv(paste0(path, "event0000010", ii, "-particles.csv"),
header=TRUE, sep=","))

  data = hits
  data = merge(data, truth, by="hit_id")
  data = merge(data, particles, by="particle_id")
  data[, rv:=sqrt(vx*vx + vy*vy)]
  data = data[rv <= 1 & vz <= 50 & vz >=-50,]
  data = data[weight > 0]
  data[, event_id:=1000+ii]
  data[, pt:=sqrt(px*px+py*py)]
  data[, alpha:=exp(-8.115 - log(pt))]

  if (ii == 0) {
    helix_params = as.data.frame(data[, .(event_id, particle_id, alpha, vz)])
  } else {
    helix_params = rbind(helix_params, as.data.frame(data[, .(event_id, particle_id, alpha,
vz)]))
  }
}

df = helix_params %>% group_by(event_id, particle_id) %>%   filter(row_number()==1)
df <- na.omit(df)
```

5.3.2 Removing outlier clusters

This approach added a postprocessing step to the algorithm. It attempted to remove unrealistic tracks, by using a least-squares regression to a cylindrical surface to estimate the fit quality of the track. Tracks which did not have a good fit were removed. However, this resulted in a slightly lower score of approximately 0.45.

```
find_norms <- function(hits, min, max) {
  hits[, track_id:=ifelse(N1 < min | N1 > max, 0, track_id)]
  hits[, nm:=0]

  clusters <- unique(hits$track_id)

  counter = 0
  for (cluster_id in clusters) {
    print(paste0("Analysing cluster ", counter, " out of ", length(clusters)))
    cluster <- hits[which(track_id==cluster_id),]
    size <- nrow(cluster)
    counter <- counter + 1

    if (cluster_id==0) next

    xyz <- as.matrix(cluster[, .(x, y, z)])
    Z <- matrix(nrow=size, ncol=10)
    x <- xyz[,1]; y <- xyz[, 2]; z <- xyz[, 3];

    Z[, 1] <- x*x
    Z[, 2] <- 2*x*y
    Z[, 3] <- 2*x*z
    Z[, 4] <- 2*x
    Z[, 5] <- y*y
    Z[, 6] <- 2*y*z
    Z[, 7] <- 2*y
    Z[, 8] <- z*z
    Z[, 9] <- 2*z
    Z[, 10] <- 1

    sv <- svd(Z, nu=nrow(Z), nv=ncol(Z))
    min_index = which.min(sv$d)
    T <- as.matrix(sv$v[min_index,])
    fit <- norm(Z %*% T, type="2")**2
    hits[, nm:=ifelse(track_id==cluster_id, fit, nm)]
  }
```

6 Conclusions

The clustering approach clearly has potential. The behaviour of the clustering approximates the behaviour of the ground truth, and requires no training time. However, looking at the leaderboard, supervised learning approaches outperformed it considerably, with the leader reaching scores of 0.92. Given more time, I would have implemented an approach involving a supervised evaluation of the tracks. Still, given this model's simplicity, it is an impressive score.

This dataset, and the approaches it received, have been extremely useful in developing new algorithms for track reconstruction. The next phase of the competition, increasing the speed of the approach, drew considerable interest, combining the already high accuracy of the leading approaches with massively reduced running times.

7 References

[1] Kaggle, "Kaggle," 2018. [Online]. Available: https://www.kaggle.com/c/trackml-particle-identification/overview/timeline. [Accessed 14 April 2020].

[2] Y. Reina, "Kaggle," [Online]. Available: https://www.kaggle.com/yuval6967/7th-place-clustering-extending-ml-merging-0-75#Chapter-3:-Expand-tracks. [Accessed 15 06 2020].

[3] V. Gaitan, "Kaggle," 2018. [Online]. Available: https://www.kaggle.com/vicensgaitan/r-scoring-function. [Accessed 10 06 2020].

[4] "Wikipedia," [Online]. Available: https://en.wikipedia.org/wiki/DBSCAN. [Accessed 14 06 2020].

[5] D. Rousseau, S. Amrouche, P. Calafiura, V. Estrade, S. Farrell and e. al, "NIPS 2018 Competition proposal: The TrackML challenge," 2018.

[6] S. Amrouche, L. Basara and a. et., "The Tracking Machine Learning challenge," *NeurIPS 2018,* 2018.