# Developing a machine learning classifier for particle ID

Michael Z Reynolds — PHYS8001 Term Project — Spring 2021

## 1 Introduction

In particle collisions, many particles are created in the process. Depending on the size and energy of the collision system, this could be hundreds or thousands of particle tracks in a wide variety of particle types and flavors. Very often, a particle will decay into a dilepton pair of opposite charge. There are three generations of leptons: electron, muon, and tau. Each have an associated neutrino. Tau's do not occur that often because they are too heavy and thus are not stable. Many particle's can be identified by their decay into dileptons. However, it is impossible to know which leptons came from which event. Because of this, there is a method for subtracting out the so-called "combinatorial background."

In the like-sign pair method, the combinations of $l^+l^+$ and $l^-l^-$ are always uncorrelated. This is because dimuon pairs are always created as $l^+l^-$. The total number of ways to combine $N$ total pairs is $N^2$ possible combinations. Which leads to $N^2 - N$ uncorrelated pairs. Thus:

$$N^2 - N = N^{++} + N^{--} \tag{1}$$

And the signal, S, is related to:

$$S^{+-} = N = N^{+-} - (N^{++} + N^{--}) \tag{2}$$

While the background, B, is related to:

$$B^{+-} = N^2 - N \tag{3}$$

In the mixed-event method, an $l^+$ and an $l^-$ are taken from different events (e.g, one from a decay of a $J/\psi$ and another from an $\Upsilon$), and a similar procedure is followed. These relations will allow us to construct the entire spectrum of dilepton pairs, and once subtracted out, particles will appear as spikes centered on its mass. Machine learning could be a useful way to classify particles, and study through simulation many collisions to make predictions about the outcomes.

### 1.1 The data

The data set I've chosen was referenced in our course material [2], and was originally published in a paper [1] where the authors built a deep learning NN to look for new physics processes in the data. It is a large set of MC simulations where particles decay into dilepton or semileptonic decays, and used by the authors as a deep learning analysis for a super-symmetry (SUSY) model. In this case, the authors are looking for super-symmetric particles through missing energy. If all the particles are accounted for, and there is still a bit of energy missing, then this could indicate that a new type of particle could have carried it away.

In this data set there are a total of 18 features: 8 low-level (raw) features, which are mostly kinematic in nature, and 10 high-level features which are functions of the raw features. These high level features are relevant to the authors' efforts, but are useless to me. So for my analysis I only used no more than the 8 raw features. These features are for each lepton transverse momentum ($p_T$), pseudo-rapidity ($\eta$), azimuthal angle ($\phi$), and then two for the missing energy (magnitude and $p_T$).

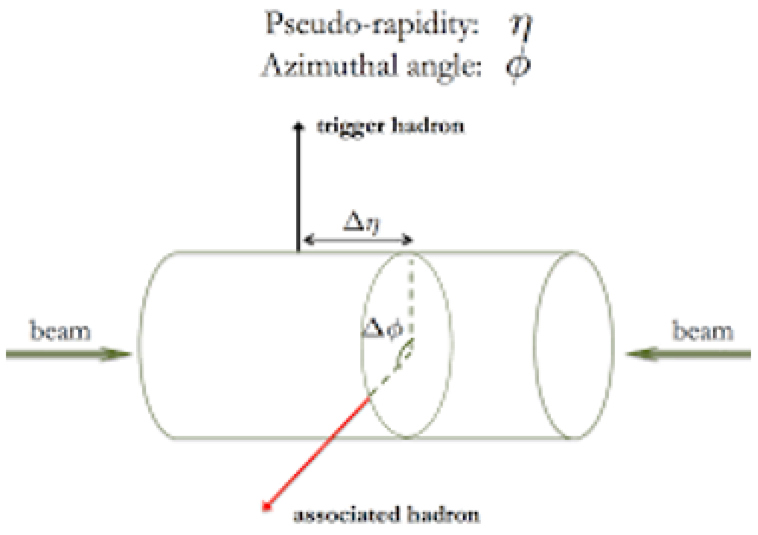

Figure 1: By convention, the beam direction is the z-axis and the red arrow would represent $p_T$. Pseudo-rapidity is the angle between the particle track and the beam direction, but it isn't quite clear in the diagram.

Each low-level feature has its own distribution which can be seen in Fig 2. It took me a bit of time to realize this because at first I was making them into scatter plots, which did not look right. My goal with this project was to build a relatively simple classifier to look at what particles are created in the collision. The data also came with labels, either signal (1) or background (2). These were used by the authors to predict if the event was a signal (missing energy) or background (any known process). I ended up attempting to use these too.

I also applied some regularization (Ridge and Lasso), but did not get the chance to pursue it further. The plots are in my Jupyter notebook. It is also possible that since each feature is itself a distribution, that I could apply some MCMC algorithms to them to make predictions about the probability of a particle appearing in the distribution. I did not have the time to pursue these avenues in the project, but at least wanted to mention that I thought about it.

Finally, I'd like to point out that in all instances of machine learning applied to the data I am using $n = 10,000$ samples, with the exception of one run that is discussed in that section.

The units for all $pT$ and mass plots are GeV.

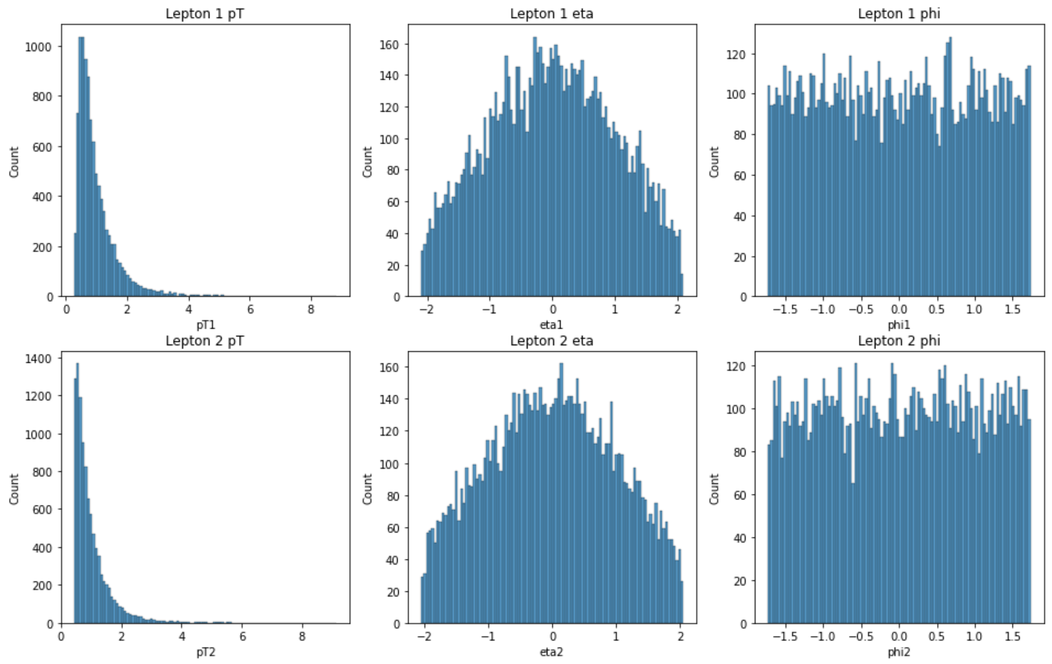

Figure 2: Each feature has it's own distribution. Units for $p_T$ are in GeV.

## 2 Methods

My first objective was to create the total invariant mass distribution of a sample from the data set, which can be seen in Fig 2. The equations I used required some assumptions, and this was to keep things simpler. The primary assumption is that the leptons are relativistic, and at these energies I could, with a small stretch, call them as such. So with $E >> m$, we have:

$$m_{inv} = 2p_{T1}p_{T2}(\cosh(\eta_1 - \eta_2) - cos(\phi_1 - \phi_2)) \tag{4}$$

Given that this equation has all the variables in my data set, I thought it would be a good fit. So I tried it, and Fig 3 was the result. This looks as I would expect, with most of the particles in the low $p_T$ range and dropping off exponentially for higher transverse momentum. The next step was to create another invariant mass distribution, but this time from **mixed events**. The idea is that one lepton from our target event (particle mass) and the other particle from an uncorrelated event. This then becomes the combinatorial background to be subtracted.

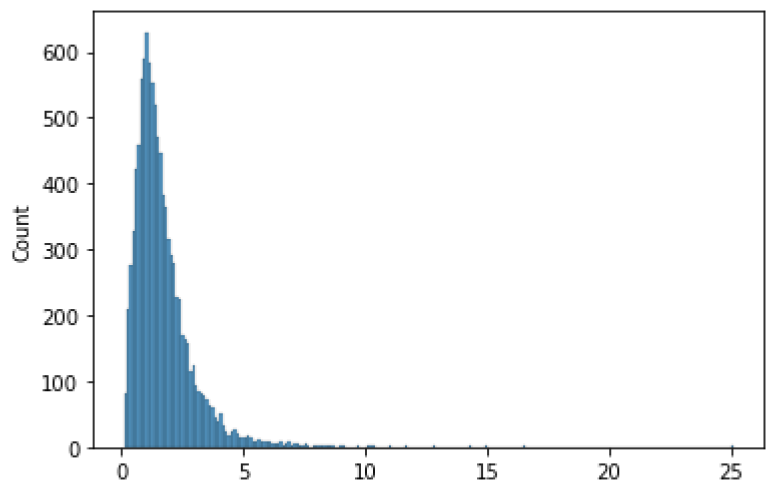

Figure 3: This is the combined invariant mass based on the raw features of each lepton. Mass units are in GeV.

My second objective was to see if I could used supervised learning to predict the label on a test sample of the data. Since the data came labeled I thought there could be a pattern with which to make predictions. For this, I used several methods of increasing complexity, starting out with the SciKitLearn tools then moving on to a SoftMax multinomial regression. Finally, I tried using a simple decision tree.

## 3 Results

For the first objective, the background subtraction procedure did not work. The result can be seen in Fig 4, where it is immediately clear that something is wrong. The peak is centered at zero, which makes no sense since this is a mass distribution. Furthermore the peak goes into negative values, which is also not possible for a mass spectrum. It was at this point that I realized this method is more complex than I initially expected, and that I did not have enough information to even attempt. First issue is that these particles are not ID'd, that is we don't know if they are electrons, muons, or neutrinos. This is an issue because the masses are vastly different, and the ID informs us of the kind of particle from which they decay. The masses also are an important part of the total energy, which is needed to properly calculate the background. And finally, the charge of the lepton is also unknown. Both the like-sign method and mixed-event method really needs the charge to properly construct the combinatorial background. In order to make the calculations, I made the assumption that the positive and negative charges are evenly distributed. But in reality, there are semi-leptonic decays that would create an imbalance in the number of pairs and the distribution of charges. As a result of these things, the spectrum left after background subtraction doesn't represent anything physical.

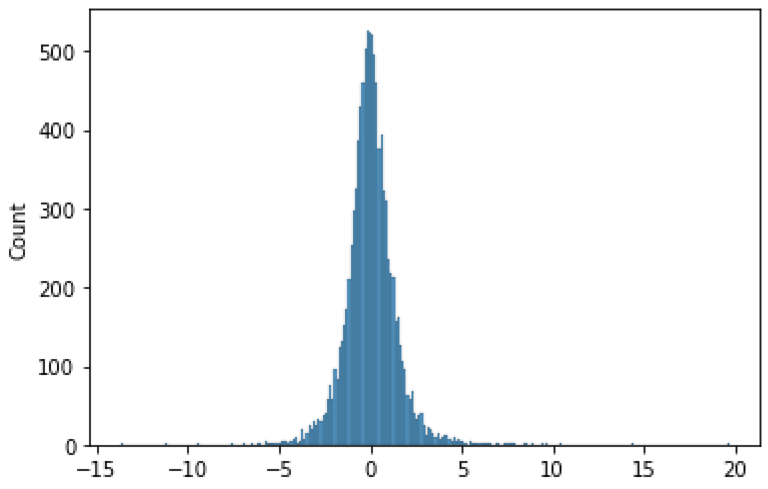

Figure 4: This peak cannot represent a physical mass. Units are GeV.

In the second objective, I started with simple classifiers built-in to the SKLearn module: LogisticRegression(), KNeighborsClassifier(), GaussianNB(), LinearSVC(), DecisionTreeClassifier(), and RandomForestClassifier(). I ran them all twice–once with all (low-level) features, and once with only $p_T$ and $\eta$. In both cases, see table 1, the predictions are better than random chance, but not very good at making confident predictions.

Table 1: SKLearn classifiers predicting signal or background labels

| **Classifier** | **Accuracy** (all features) | **Accuracy** ($p_T$ and $\eta$) |
|---|---|---|
| Logistic Regression | 77.7 | 72.3 |
| KNeighbors | 74.0 | 68.8 |
| Naive Bayes | 74.2 | 66.7 |
| Linear SVM | 77.3 | 71.5 |
| Decision Tree | 69.4 | 62.4 |
| Random Forest | 78.7 | 70.9 |

Ultimately, I thought this was too simple, as it is only a few lines of code. So I next tried some of the classifiers we used in the course assignments. The first was the Softmax multinomial logistic regression using PyTorch (week 6 assignment). However, I kept getting an error (ValueError: too many values to unpack (expected 2)), which I believe is related to the features (input). The original code was for an image classifier, which has very different features than the SUSY data. I did some quick troubleshooting, but in the interest of time had to move on to another method. I then tried the CART decision tree (week 3). This was a fairly simple bit of code, and took a few minutes to run each time. Turns out the data set is way too complex for the decision trees to be useful. See Figs 5 and 6 for a visualization of these trees. The second set of trees is a much smaller sample size of only 1000 events.

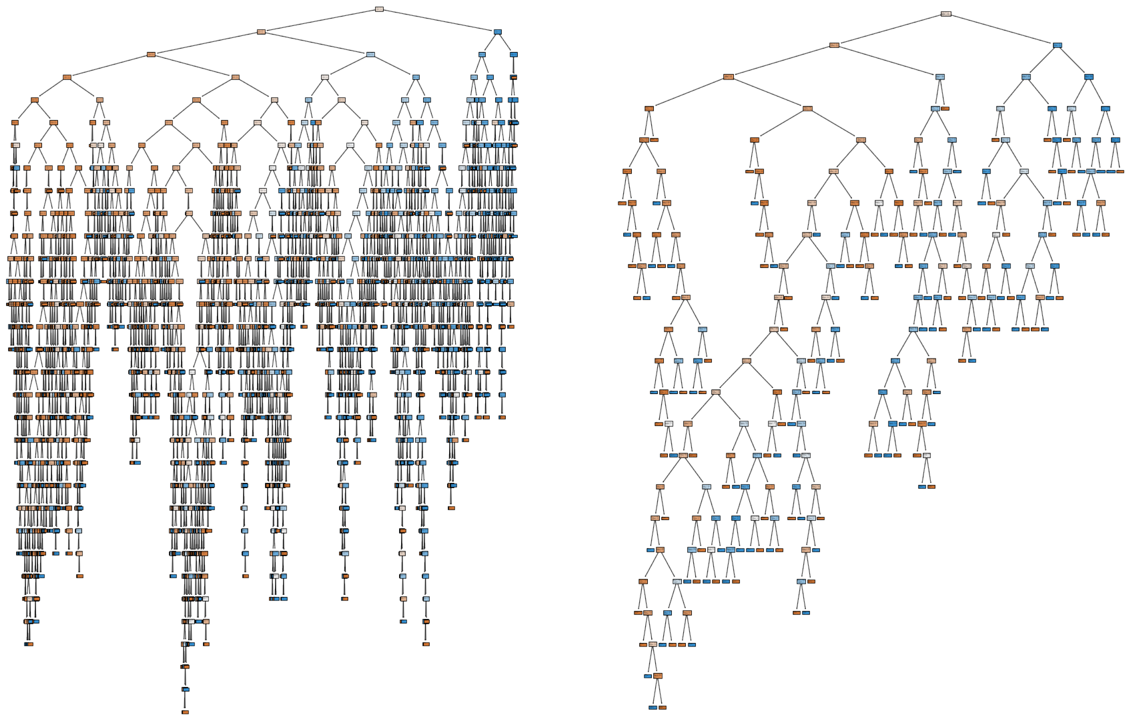

Figure 5: This is with the 'gini' classifier with 10k events split into 90% training (left) and 10% testing (right). Using 'entropy' instead produced a similar result.

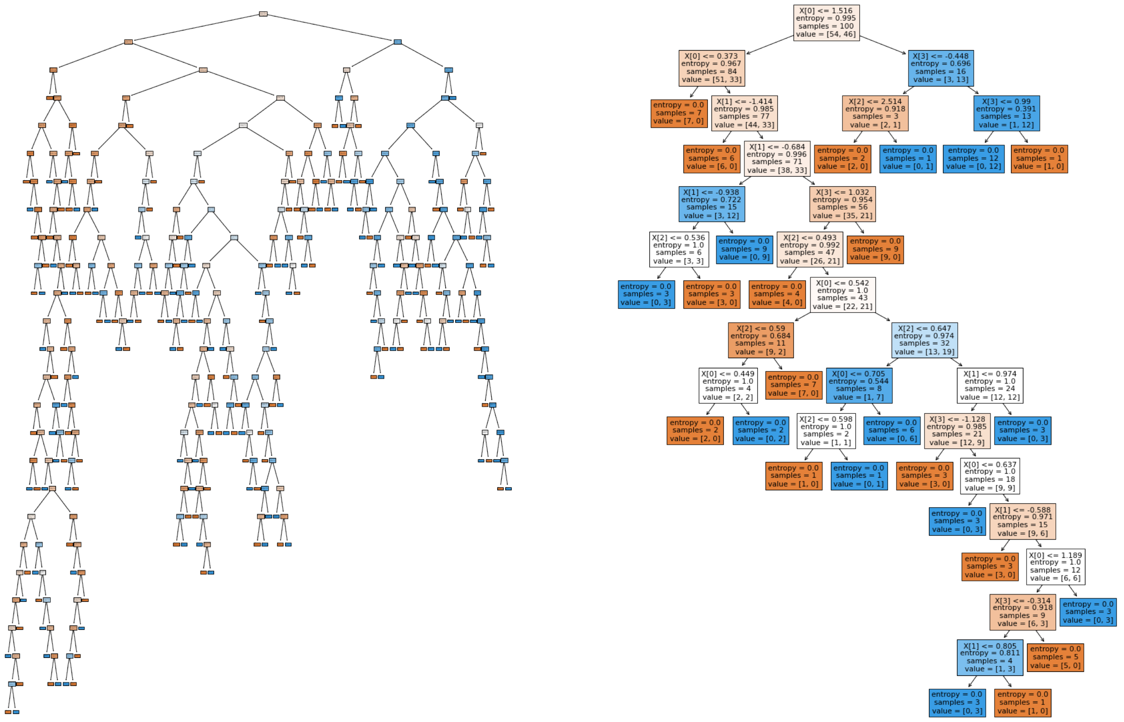
X[0] <= 1.516 entropy = 0.995 samples = 100 value = [54, 46]
X[0] <= 0.373 entropy = 0.967 samples = 84 value = [51, 33]
X[3] <= -0.448 entropy = 0.696 samples = 16 value = [3, 13]
entropy = 0.0 samples = 7 value = [7, 0]
X[1] <= -1.414 entropy = 0.985 samples = 77 value = [44, 33]
X[2] <= 2.514 entropy = 0.918 samples = 3 value = [2, 1]
X[3] <= 0.99 entropy = 0.391 samples = 13 value = [1, 12]
entropy = 0.0 samples = 6 value = [6, 0]
X[1] <= -0.684 entropy = 0.996 samples = 71 value = [38, 33]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 12 value = [0, 12]
entropy = 0.0 samples = 1 value = [1, 0]
X[1] <= -0.938 entropy = 0.722 samples = 15 value = [3, 12]
X[3] <= 1.032 entropy = 0.954 samples = 56 value = [35, 21]
X[2] <= 0.536 entropy = 1.0 samples = 6 value = [3, 3]
entropy = 0.0 samples = 9 value = [0, 9]
X[2] <= 0.493 entropy = 0.992 samples = 47 value = [26, 21]
entropy = 0.0 samples = 9 value = [9, 0]
entropy = 0.0 samples = 3 value = [0, 3]
entropy = 0.0 samples = 3 value = [3, 0]
entropy = 0.0 samples = 4 value = [4, 0]
X[0] <= 0.542 entropy = 1.0 samples = 43 value = [22, 21]
X[2] <= 0.59 entropy = 0.684 samples = 11 value = [9, 2]
X[2] <= 0.647 entropy = 0.974 samples = 32 value = [13, 19]
X[0] <= 0.449 entropy = 1.0 samples = 4 value = [2, 2]
entropy = 0.0 samples = 7 value = [7, 0]
X[0] <= 0.705 entropy = 0.544 samples = 8 value = [1, 7]
X[1] <= 0.974 entropy = 1.0 samples = 24 value = [12, 12]
entropy = 0.0 samples = 2 value = [2, 0]
entropy = 0.0 samples = 2 value = [0, 2]
X[2] <= 0.598 entropy = 1.0 samples = 2 value = [1, 1]
entropy = 0.0 samples = 6 value = [0, 6]
X[3] <= -1.128 entropy = 0.985 samples = 21 value = [12, 9]
entropy = 0.0 samples = 3 value = [0, 3]
entropy = 0.0 samples = 1 value = [1, 0]
entropy = 0.0 samples = 1 value = [0, 1]
entropy = 0.0 samples = 3 value = [3, 0]
X[0] <= 0.637 entropy = 1.0 samples = 18 value = [9, 9]
entropy = 0.0 samples = 3 value = [0, 3]
X[1] <= -0.588 entropy = 0.971 samples = 15 value = [9, 6]
entropy = 0.0 samples = 3 value = [3, 0]
X[0] <= 1.189 entropy = 1.0 samples = 12 value = [6, 6]
X[3] <= -0.314 entropy = 0.918 samples = 9 value = [6, 3]
entropy = 0.0 samples = 3 value = [0, 3]
X[1] <= 0.805 entropy = 0.811 samples = 4 value = [1, 3]
entropy = 0.0 samples = 5 value = [5, 0]
entropy = 0.0 samples = 3 value = [0, 3]
entropy = 0.0 samples = 1 value = [1, 0]

Figure 6: This is with the 'entropy' classifier with 1k events split into 90% training (left) and 10% testing (right).

## 4 Summary

In this project, I attempted to code a classifier for particles created in simulated proton-proton collision data. I naively believed that I could do this strictly with the raw (low-level) data. It may have been possible if I had more raw data information, such as charge and particle ID. However, with what I had I needed to make too many assumptions, which made the problem trivial. In order to properly build this classifier I would need to make my own high-level features that use the raw data in ways useful to the classification. Then I tried to use premade classifiers (from class), but they were too simple for the data set. So I found myself with the opposite problem: in my first objective the data was too simple for the classifier, and in my second objective the classifier was too simple for the data. This is a project that I am actually interested in pursuing outside of class, and as I progress in my dissertation (which is an analysis on a very similar data set) I will gain more insights on how to build this classifier.

## References

[1] Baldi, P., P. Sadowski, and D. Whiteson. “Searching for Exotic Particles in High-energy Physics with Deep Learning.” Nature Communications 5 (July 2, 2014) doi:10.1016/j.physletb.2019.02.018

[2] Mehta, P., M Bukov, et al. ”A high-bias, low-variance introduction to Machine Learning for physicists.” Physics Reports 810 (May, 2019) doi.org/10.1016/j.physrep.2019.03.001