JetML: Finding Jets in Heavy Ion Particle Collisions with Machine Learning Algorithms

Mark Kocherovsky
Lawrence Technological University, Department of Mathematics and Computer Science, Southfield, MI 48075
(Dated: 18 Dec 2020)

Finding instances of "jets" in simulated relativistic heavy-ion collision events is presently done by checking the momenta of particles produced in the collision. This paper explores an alternate method of tagging events with jets using the event-level data alone, without having to check specific patterns in the particle-level data. Six basic machine learning algorithms are tested on their ability to find events with jets within four datasets, where an event with at least one jet is classified as "Jet," and an event without jets is classified as "No Jets." It is found that such a method is viable, and that the decision tree classifier algorithm can determine the presence of at least one jet with 100% accuracy in a short-enough amount of time to be practical. The program can also generate graphs visualizing event data and the effectiveness of each algorithm, as well as output each model for future use. Although the program, as of yet, does not currently derive the *amount* of jets in each event, the experiment shows that it is possible to use machine learning to detect the presence of jets using data from relativistic heavy-ion collisions.

I. BACKGROUND

A. Physics Background

When two particles (such as two protons or two atomic nuclei) collide at relativistic (ultrahigh) velocities, a slew of new particles are created. It is thought that there are two methods by which particles are created. One of these methods is "thermal" production, where particles are formed out of cooling quark-gluon plasma that is created by the collision. The other possible method is a "jet" of particles that is caused by a kicked out quark or gluon (the fields between the quarks will break, creating a shower of particles). Neither thermal nor jet production can be individually detected by instruments; experiments can only detect particles as they come into contact with the detector. Therefore, the existence of jets must be determined mathematically from properties of the resultant particles [1]. Jets are defined using the following properties:

- Jet cone radius (in momentum space) $R = 0.4$.
- Total jet momentum $P_T > 5\ GeV$, where P_T is event total momentum in the transverse – perpendicular to the particle beam – direction.
- Leading particle $p_T > 8\ GeV$, where p_t is the momentum in the transverse direction for an individual particle.
- Sub-leading particle $p_T > 5\ GeV$.
- Constituent particles have $0 < p_T < 1\ GeV$.
- Back-to-back jets are required, with a tolerance of 0.4 radians from π.

Figure 1 shows a diagram of jets in a proton-proton collision [1].

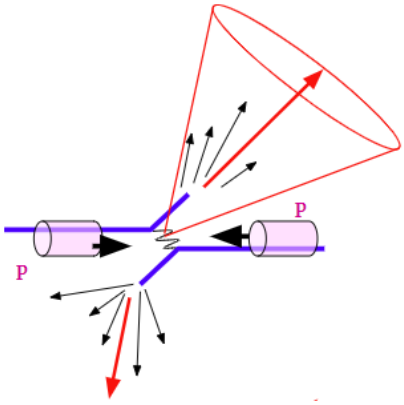

FIG. 1. A diagram of jets in a proton-proton collision. The red arrow indicates the leading particle, and the black arrows indicate constituent particles. The cone represents the jet in momentum space.

Collisions can be simulated using PYTHIA, developed at Lund University. PYTHIA enables researchers to simulate collisions between various particles at various levels of energy, and collect data from the simulated collision events [2]. The data can then be organized, read, and presented using the ROOT Data Analysis Framework for C++ [3].

B. Machine Learning Background

Machine learning is a field of study focusing on algorithms capable of learning from experience [4]. Though there are several methods and algorithms to perform machine learning, the JetML program tests and compares six classification (assigning a category based on input data) algorithms. **Logistic regression** (LR) uses a logistic function that linearly combines the given data and predicts the probability of that data resulting in a de-

sired outcome [5]. By comparison, a **linear discriminant** (LDA) function is a specifically linear combination specifically designed to produce a classification [6]. The **k-nearest neighbor** (KNN) algorithm divides a set of data (represented on a scatter plot) into several subsets called "neighborhoods." The algorithm will then calculate local means, variances, etc., and output classifications based on these values [7]. **Decision tree classifiers** (DTC) build a set of paths in a tree format that allows the program to assign a classification using a set of sequential decisions (hence the name "decision tree") [8]. The **Gaussian naive Bayes** (GNB) algorithm uses the Bayes theorem to estimate a Gaussian distribution for classification [9]. Finally, **support vector machines** (SVM) map inputs to a multi-dimensional vector space, from which a linear surface is constructed. This allows classification [10].

C. Purpose of the JetML Program

The purpose of the JetML program is to test the aforementioned machine learning algorithms to compare each algorithm's ability to use collision event data to predict whether or not that event contains a jet. Ability is measured using a built-in accuracy measurement in the Python scikit-learn library (see Section II B). In addition, the algorithms should be able to perform the calculations quickly in order to be practical.

II. PROGRAM REQUIREMENTS

This section aims to explain what the JetML program is meant to accomplish and what it is meant to output to the user. For a more in-depth explanation of how the program operates, please see Section III.

A. Organizing the Data

Before applying machine learning, the data must be preprocessed. This was to be done with a program (called `EventCataloguer.cxx`) on our research team's shared computer. The event cataloguer program took quantitative data at the event level (see Section IV A) and outputted them into a comma-separated values (.csv) file. In addition, the program would mark down whether or not the event contained jets (labeled as "Jet" or "No Jets"). Once complete, the file can then be read by the JetML program proper.

B. The JetML Program

The `JetML.py` program is meant to ascertain whether or not basic machine learning algorithms could accurately predict if an event contain jets or not. The program tests logistic regression, linear discriminant analysis, k-nearest neighbors, decision tree classifier, Gaussian naive Bayes, and support vector machine algorithms. Each of these models are found in the scikit-learn (`sklearn`) Python library [11]. The models are trained on half the data, and then tested on the other half. JetML then returns an accuracy score between 0.0 and 1.0. A higher score indicates a more accurate algorithm. Secondary objectives included time performance, as an algorithm that is too slow loses practical value; and graphical representation of the results, in the form of a bar graph comparing the various accuracy scores, a plot showing which events do or do not have jets in relation to the numerical data, and precision-recall curves (precision is a measure of the algorithms' ability to avoid false positives, and recall is a measure of the algorithms' ability to avoid false negatives [12]) for each algorithm. Appendix A shows examples of each graph. Finally, the program will output the trained models for future use. A graphical representation of the requirements is shown in Figure (2).

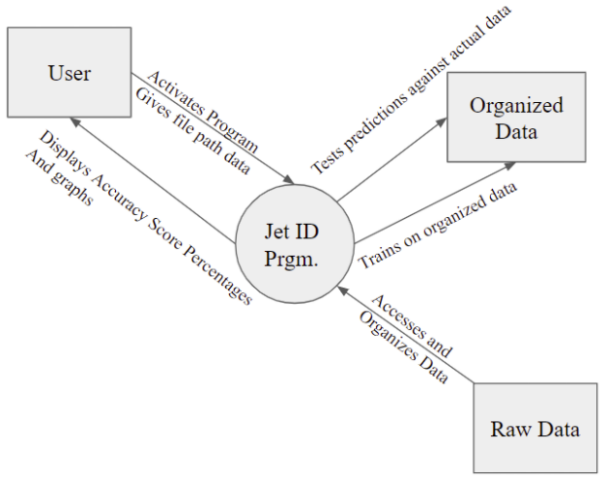

FIG. 2. System Context Diagram for the JetML program.

III. DESIGN

The preprocessing program, `EventCataloguer.cxx`, is built in C++. It opens each data file and loops through each event in series, writing the relevant data to a `.csv` file.

The `JetML` program is a Python program that takes the data file and imports the data into a table. The data is then split into the features (input data) and label (resultant classifier). The program then sequentially splits the input data into testing and training sets (the proportion is given by the `split_proportion` variable, set to 0.5 by default) in the `JetML` function. The function will then perform training, and then output the trained model as a `.joblib` file. It will then perform predictions on the testing set and measure the accuracy (a score of 1.0 in-

dicates that the model is accurate 100% of the time). Finally, the function measures and records how long the model took to train and test, and outputs the relevant data as lists. The program will then write these results to a text file as output.

`JetML.py` then moves on to creating graphical output. First, it creates a bar graph showing the accuracy score of each algorithm. It will also label each bar with the numeric value, and save the graph as `results.png`.

The second graph is the precision-recall comparison. The labels are at first normalized, so that if the input data has a jet, then it will be marked as 1 in the array, and otherwise will be marked as 0. The program then makes predictions and calculates the precision-recall curve, and plots it on the graph. The program does this for each model, enabling a comparison of each model. This is then plotted with a no-skill model value (a no-skill model essentially is the probability of correctly assigning the desired label to events with jets at random [13]. The precision-recall graph is saved as `precreccurve.png`.

Finally, `JetML.py` creates a scatter plot using the input features. Though the user may choose to change the code to plot different features on the axes, the program currently plots `PTtot` against `Ntot` (see Section IV A). Each data point is colored depending on the label. Events without jets are marked in red, while events with at least one jet are marked in green. This graph – or variations of it – can be used to infer how the various measurements influence the final classification. This is saved as `jets_nojets.png`. All other functionality is command-line based.

A diagram of this design can be seen in Figure (3).

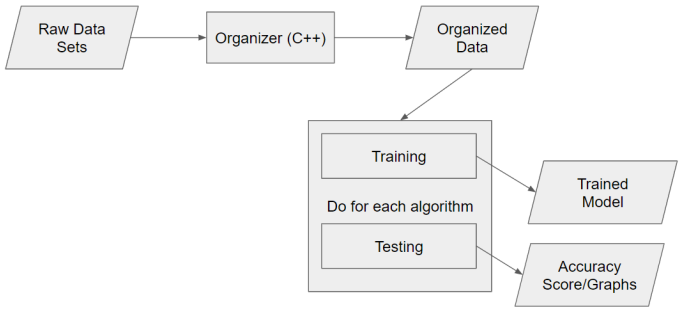

FIG. 3. Flowchart for the JetML program.

IV. DATASETS AND RESULTS

A. Datasets

JetML was tested on four datasets. Each dataset contained several events, each with sixteen measurements:

- `Mult`: the total number of all particles created in the collision, including those that themselves form or separate into new particles.
- `Ntot`: the end total number of particles that pass through filtering meant to mirror the limitations of a real detector.
- `Nhard`: the end total number of "hard scatterings" (large momentum transfers in the movement of particles [14]).
- `Nch`: the end total number of electrically charged particles in `Ntot`.
- `Nplus`: the end total number of particles with positive charge.
- `Nminus`: the end total number of particles with negative charge.
- `E_tot`: the total energy of all particles in `mult` (will be constant within each dataset, measured in GeV).
- `Ech_tot`: the end total energy of all charged particles (measured in GeV).
- `ET_tot`: the total energy of particles in the transverse direction (measured in GeV).
- `ETch_tot`: the total transverse energy of all charged particles (measured in GeV).
- `PT_tot`: the end total momentum of all charged particles in the transverse direction (measured in GeV/c).
- `PTch_tot`: the end total momentum of all charged particles in the transverse direction (measured in GeV/c).
- `sqrts` ($\sqrt{s}$): the center-of-mass collision energy per nucleon (a proton or a neutron in a nucleus). This will be constant within each dataset as well (measured in GeV).
- `Ncoll`: the total number of sub-collisions between separate nucleons.
- `Npart`: The total number of wounded nucleons (have interacted with others at least once [15]).
- `b`: Impact parameter (distance between the centers of the involved nuclei, measured in fm).
- Finally, there is an indication of "Jet" or "No Jets" in the event as a string.

JetML was tested on four datasets. Two were "full" datasets. The first, `PbPb2760GeVcataloguedEvents.csv`, consists of three million lead-lead collision events with $\sqrt{s} = 2760\ GeV$. The second, `pp200GeVcataloguedEvents.csv`, consists of thirty million proton-proton collision events with $\sqrt{s} = 200\ GeV$. The other two were partial datasets. `PbPb2760GeVcataloguedEvents_small.csv` contains

the first ten thousand events from its full counterpart, while `pp200GeVcataloguedEvents_small.csv` consists of the first three million events from the full proton-proton dataset. The former was created as a partial set mainly to facilitate rapid testing of the Event Cataloguer and JetML programs, as using the full datasets takes far longer to run. The latter partial dataset was created to mirror the lead-lead partial dataset.

B. Results

Table I shows the accuracy metrics of each dataset for each algorithm, where each model trains on half of the dataset, and is then tested on the other half. Note that for all datasets other than the small PbPb2760 GeV set, support vector machine models took too long to train and test, so SVM was cut for those datasets. As Table II shows, training and testing support vector machines takes two orders of magnitude longer than the next longest (logistic regression). In addition, LR and KNN results for the pp200 GeV datasets (both full and partial) are not exactly 1.0, but they are rounded up from the ten-thousandths digit.

TABLE I. Table showing the accuracy (between 0 and 1) of each machine learning algorithm for each dataset.

Dataset	LR	LDA	KNN	DTC	GNB	SVM
PbPb2760 (small)	0.961	0.938	0.963	1.000	0.947	1.000
pp200 (small)	1.000[a]	0.999	1.000[a]	1.000	0.897	[b]
PbPb2760	0.958	0.934	0.968	1.000	0.944	[b]
pp200	1.000[a]	0.999	1.000[a]	1.000	0.903	[b]

[a] Rounded up from 0.999...
[b] Not measured due to overly large calculation times.

Table II, which shows the duration (in seconds) of each algorithm run on each dataset; Table III, which shows the no-skill model value for the full datasets; and each graph for the full datasets can be found in Appendix A.

V. CONCLUSION

My results show that it is possible to use the decision tree classifier and support vector machine algorithms to predict whether or not there are jets in a particle-particle collision event with 100% accuracy. However, as support vector machines take far longer to run than the other models, their usage is not practical, especially for large datasets. Ergo, the decision tree classifier model is the most practical of the six basic machine learning algorithms.

A. Future Considerations

There are several ways that this project can be expanded on. The JetML program could be reworked to use the individual particles detected in the event to train the model, and make predictions of how many jets were in the event – not just if there is a jet at all. In addition, the JetML program could also be expanded to use more types of machine learning algorithms. Advanced algorithms could be tested and implemented, as could a neural network to test their effectiveness and performance. In addition, it would be interesting to explore whether or not machine learning can allow classification without having to define what a jet is using the traditional algorithms beforehand.

VI. ACKNOWLEDGEMENTS

This work was supported in part by the U.S. NSF grant PHY- 1913005.

[1] M. V. Kocherovsky, The variance of high energy nuclear collisions (2020), [PowerPoint Slides].
[2] T. Sjöstrand, S. Mrenna, and P. Skands, A brief introduction to PYTHIA 8.1, Computer Physics Communications **178**, 852 (2008), [Online; accessed 13-December-2020].
[3] R. Brun and F. Rademakers, ROOT - an object oriented data analysis framework, Nucl. Inst. & Meth. in Phys. Res. **389**, 81 (1996), [Online; accessed 13-December-2020; See also 'ROOT' [software]].
[4] C. Chung, Neural networks and deep learning with Python (2020), [PowerPoint Slides].
[5] A. S. Hess and J. R. Hess, Logistic regression, Transfusion **59**, 2197 (2019), [Online; accessed 05-December-2020].
[6] A. R. Webb, K. D. Copsey, and G. Cawley, *Statistical Pattern Recognition*, 3rd ed. (John Wiley & Sons, Incorporated, 2011) pp. 20–21, [Online; accessed 05-December-2020].
[7] N. S. Altman, An introduction to kernel and nearest neighbor nonparametric regression, The American Statistician **46**, 10.1080/00031305.1992.10475879 (1991), [Online; accessed 06-December-2020].
[8] B. Kaminski, M. Jakubczyk, and P. Szufel, A framework for sensitivity analysis of decision trees, Cent Eur J Oper Res **26**, 135 (2018), [Online; accessed 06-December-2020].
[9] J. C. Griffis, J. B. Allendorfer, and J. P. Szaflarski, Voxel-based gaussian naïve bayes classification of ischemic stroke lesions in individual t1-weighted mri scans, Journal of Neuroscience Methods **257**, 97 (2016), [Online; accessed 06-December-2020].
[10] C. Cortes and V. Vapnik, Support-vector networks, Machine Learning **20**, 273 (1995), [Online; accessed 06-December-2020].
[11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cour-

napeau, M. Brucher, M. Perrot, and E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research **12**, 2825 (2011).

[12] L. P. Coelho and W. Richert, *Building Machine Learning Systems with Python - Second Edition.*, 2nd ed., Community Experience Distilled (Packt Publishing, 2015) pp. 116–117, [Online; accessed 06-December-2020].

[13] J. Brownlee, Roc curves and precision-recall curves for imbalanced classification (2020), [Online; accessed 06-December-2020].

[14] H. Frisch, Hard parton scattering (2013), [PowerPoint Slides].

[15] H. Bialkowska, Wounded nucleons, wounded quarks, and relativistic ion collisions, Acta Physica Polonica **B37** (2006), arXiv:nucl-ex/0609006 [nucl-ex].

Appendix A: Additional Data and Graphs

TABLE II. Table showing the duration (in seconds) of each machine learning algorithm for each dataset.

Dataset	LR	LDA	KNN	DTC	GNB	SVM
PbPb2760 (small)	0.286	0.087	0.268	0.081	0.032	63.19
pp200 (small)	195.7	33.52	427.2	75.22	22.50	[b]
PbPb2760	51.07	44.14	184.5	105.8	27.58	[b]
pp200	1449	634.3	[a]	1592	476.5	[b]

[a] Could not be properly measured, as the program ran for about eight hours every day, and was paused in-between.

[b] Not measured due to overly large calculation times.

TABLE III. Table of the no-skill models' values of the full datasets.

Dataset	No-skill model value
PbPb2760	0.674
pp200	9.177e-5

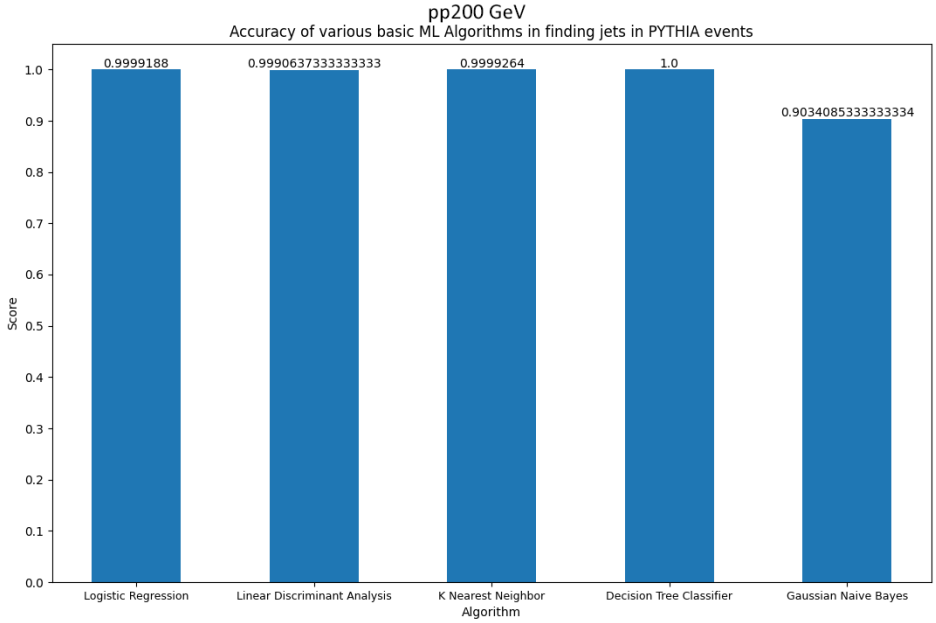

FIG. 4. Graphical comparison of machine learning model accuracy for the full pp200 GeV set.

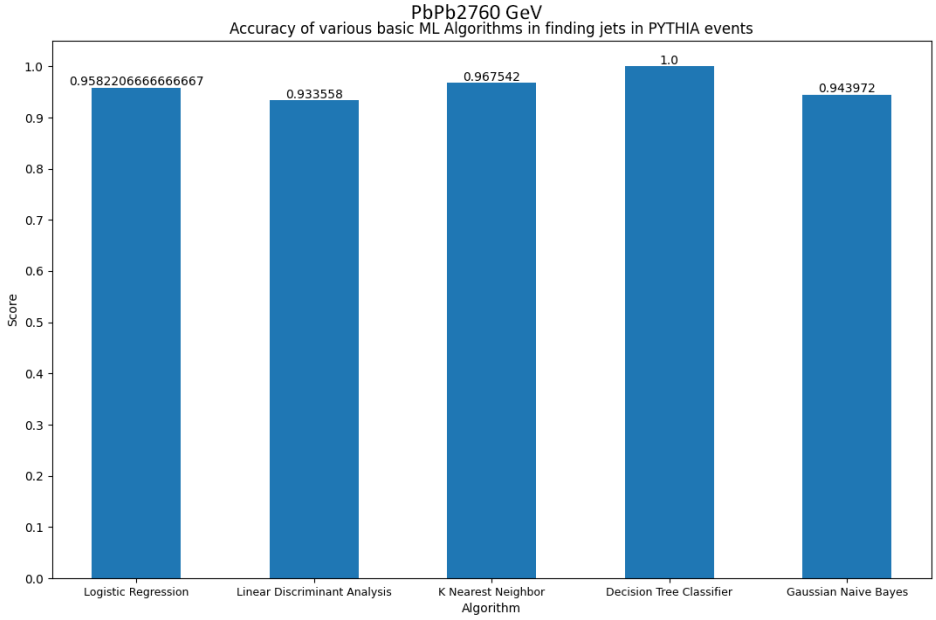

FIG. 5. Graphical comparison of machine learning model accuracy for the full PbPb2760 GeV set.

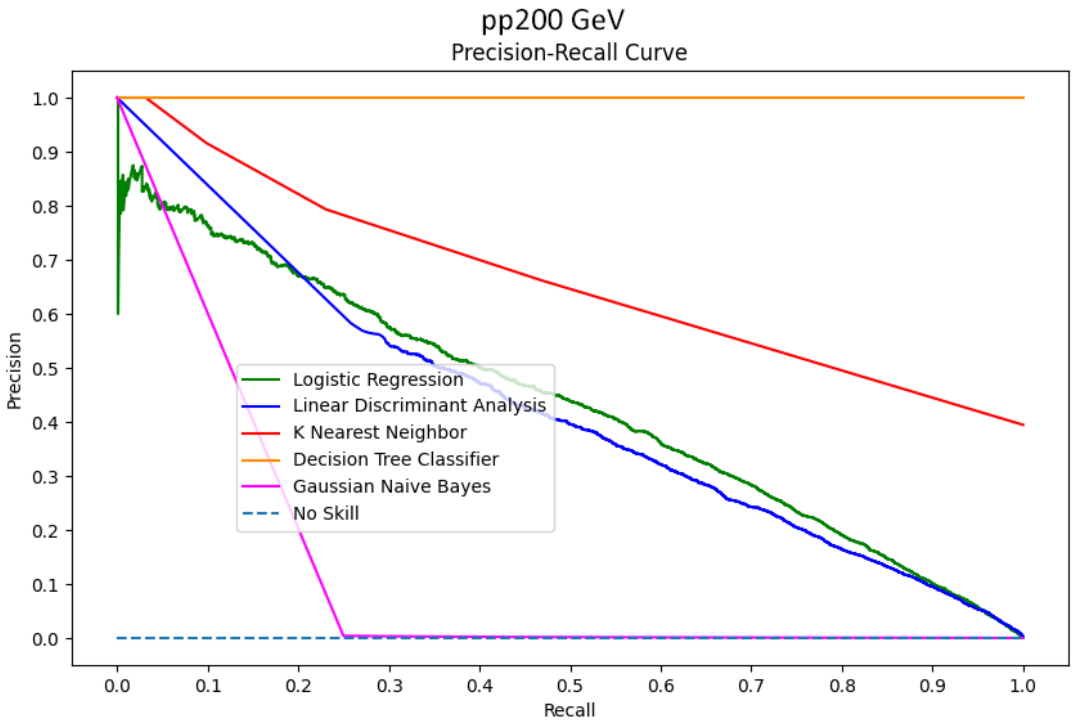

FIG. 6. Precision-Recall Curve with a no-skill value of 9.177e-5 for the full pp200 GeV dataset. Points above the no-skill line and closer to $(1.0, 1.0)$ indicate a better model.

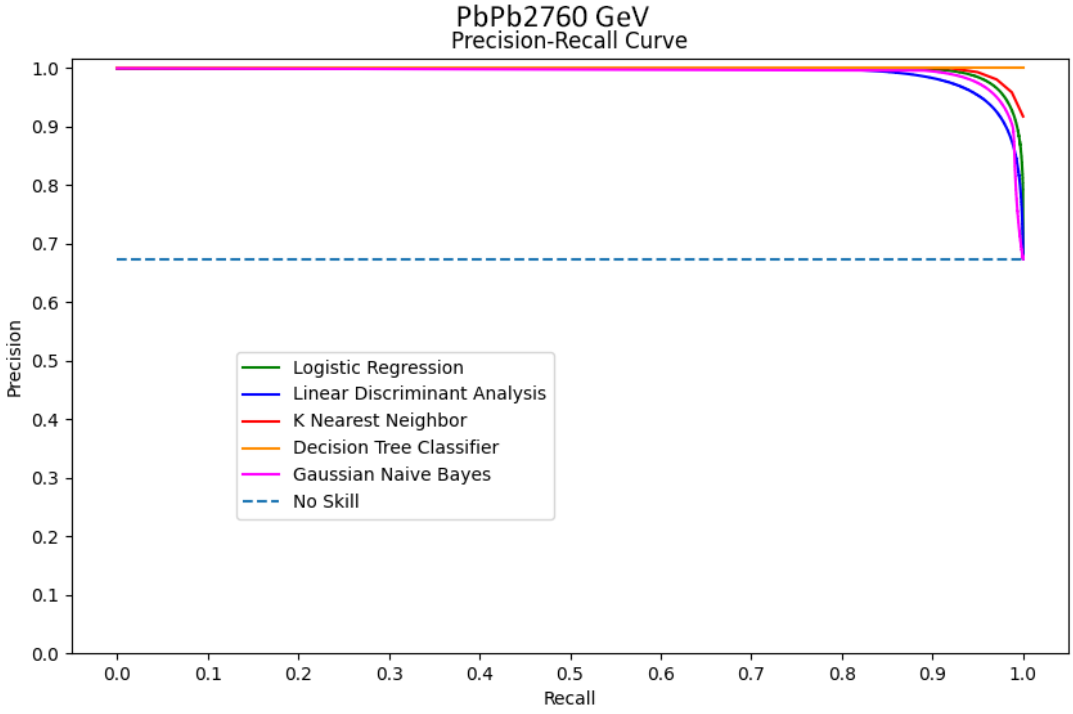

FIG. 7. Precision-Recall Curve with a no-skill value of 0.674 for the full PbPb2760 GeV dataset. Points above the no-skill line and closer to $(1.0, 1.0)$ indicate a better model.

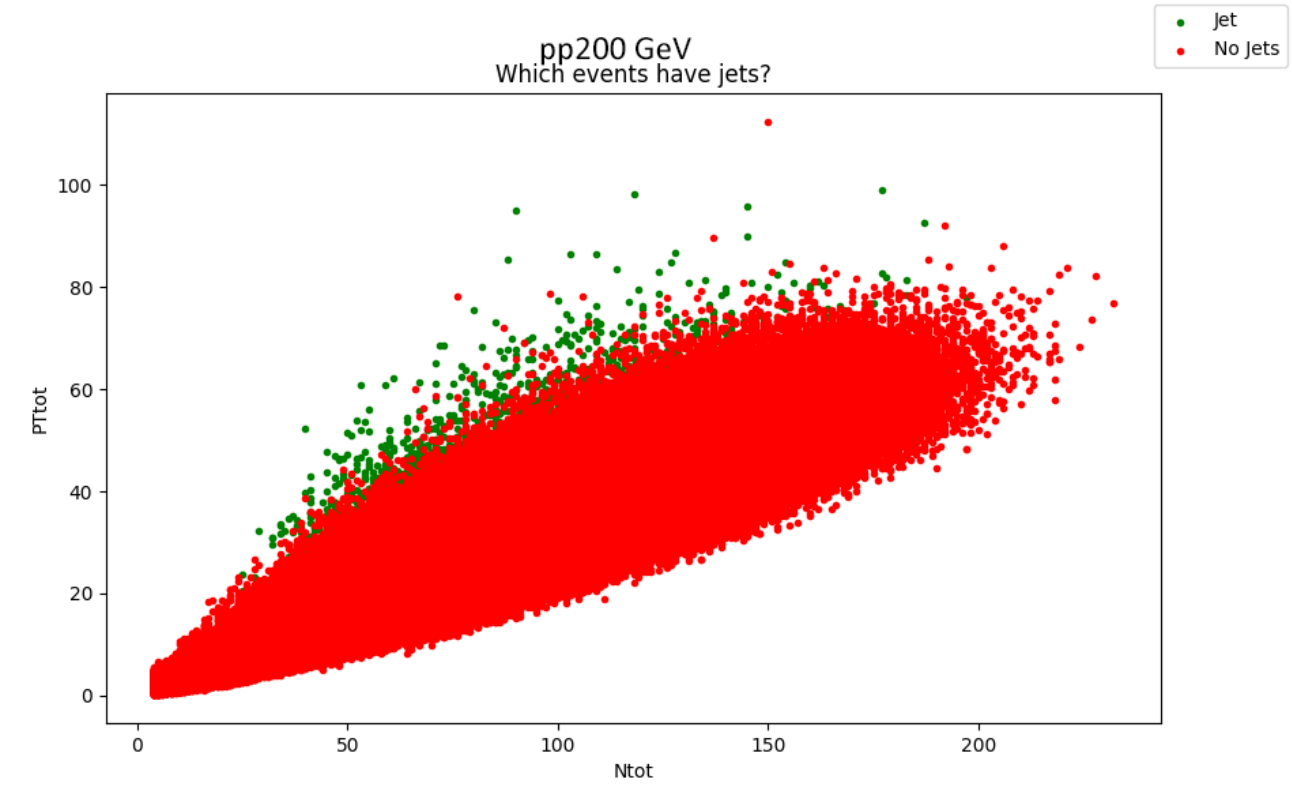

FIG. 8. A plot showing which events have or do not have jets using `PTtot` versus `Ntot` as axes for the full pp200 GeV dataset.

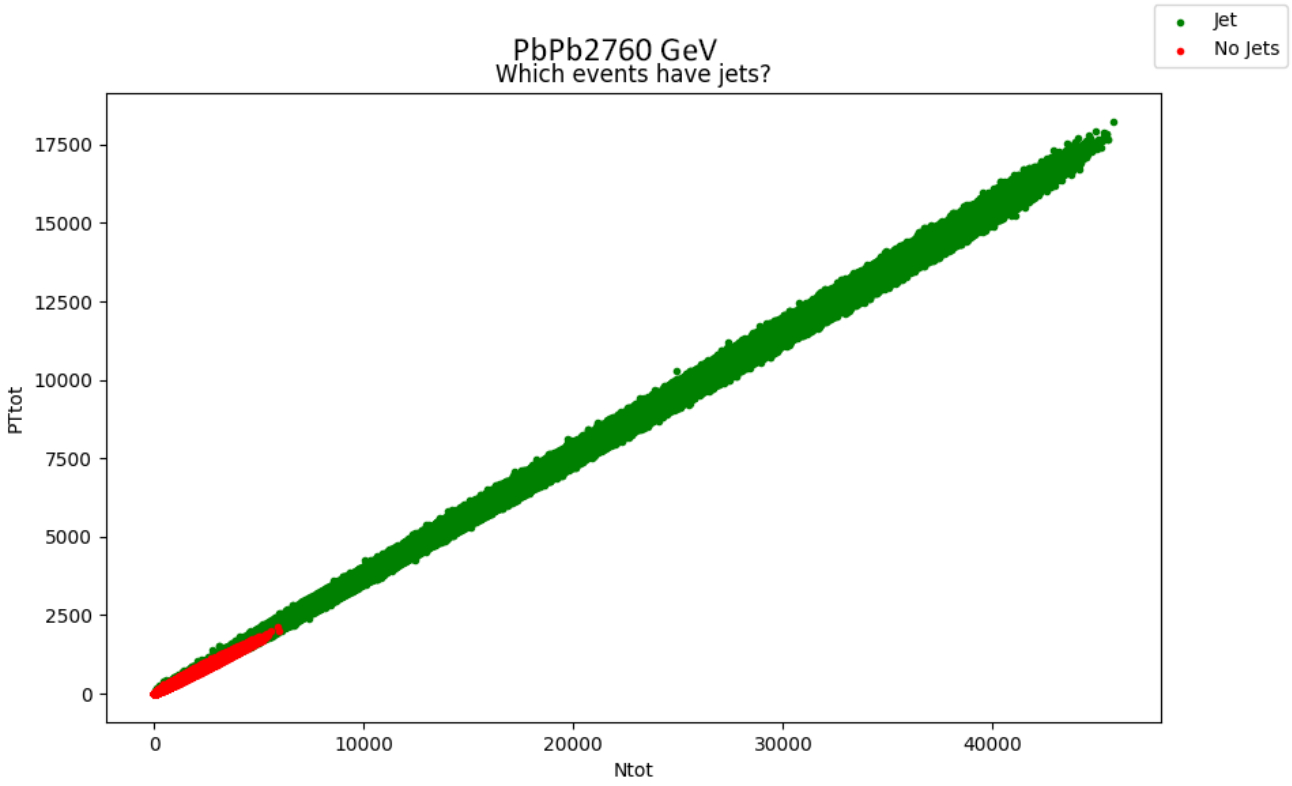

FIG. 9. A plot showing which events have or do not have jets using `PTtot` versus `Ntot` as axes for the full PbPb2760 GeV set.