Predicting Particles!

Big Data Energy

Shashank Mahesh, Emre Yurtbay

Abstract

One of the major problems in modern physics is determining the class of particles that are detected during particle accelerator experiments. Using measurement data from the Large Hadron Collider at CERN, our team built an ensemble of models to help classify whether a particle is an electron, proton, kaon, pion, muon, or a ghost particle. Our method had an accuracy of 89 percent on previously unseen test data, which suggests that this method may be very useful in helping physicists better understand results from their experiments at the LHC.

Introduction

The Large Hadron Collider (LHC) is the largest particle accelerator in the world and one of the most important tools in helping scientists make advances in modern physics. By accelerating different kinds of particles at each other at very high kinetic energies and measuring the byproduct of these high speed collisions, physicists are able to understand the structure of the subatomic world and uncover truths about the laws governing it. The way that the LHC works is that the particle accelerator sends large particles like protons around a massive circular tunnel at just below the speed of light. The collision of these particles results in high energy reactions that give rise to special conditions that are almost impossible to study in the natural world. The accelerator itself is a 16-mile tunnel on the Swiss-French border near the city of Geneva, and experiments at the LHC produce over 10 billion petabytes of data every year. Data from the LHC has been used to answer questions in areas of physics ranging from quantum mechanics to general relativity, and physicists hope that the LHC can be used to help develop a ”Grand Unified Theory” of physics. In the decade the LHC has been in operation, physicists have made many discoveries, the most important of which involves the Higgs Boson, a particle that gives other particles mass. The data set we are working with consists of 1.2 million particle accelerations, each with 49 measurements and a label describing what type of particle was sent through the accelerator. The data comes from CERN, the laboratory which houses the LHC, and was cleaned and uploaded to Kaggle. With this data, we want to try to understand the fundamental properties of different particles and see what new insights can be gleaned from the copious amounts of accelerator data. Machine learning and deep learning has been applied to LHC data in the past, for example, in ”Deep Learning and its Application to LHC Physics” by Guest et al. and ”Extending the Bump Hunt with Machine Learning” by Collins et al.

The first question we wanted to answer was a simple but important one: based on sub-detector sensor data in the LHC itself, can we build a classifier to distinguish what type of particle was detected by the accelerator? When a certain type of particle is sent through an accelerator, it leaves different traces, all of which are detected by different sub-detector sensors in the accelerator. While this seems like a simple question, it is of the utmost importance. The budget for the LHC is 9 billion dollars, making it the one of the most expensive scientific objects ever built, but even this is not enough. Getting dependable measurements inside the collider is very expensive, so machine learning may be used to lower these costs and perhaps even improve accuracy. In our data set, we have measurements from five different sub-detector systems: the tracking system, the ring imaging cherenkov (RICH) detector, the electromagnetic calorimeter (EC), and the muon chamber. The tracking system measures information about the particle's momentum, charge, and decay. The RICH detector measures particle emission angle and delta log-likelihoods for each particle type. The EC measures a particle's energy, and the Muon chamber measures whether or not the particle makes contact with it or not. Using these measurements, we want to classify each particle into one of the following categories: Electron, Proton, Kaon, Pion, Muon, or Ghost. Electrons and protons are familiar particles that make up atoms. Pions belong to a class of particles called mesons, and interact with all four fundamental forces: gravity, electromagnetism, and the strong and weak nuclear forces. Kaons are also mesons, and are important for physicists' understanding of quantum mechanics and cosmic rays. Muons belong to a class of particles called leptons, and interact with gravity, the electromagnetic force, and the weak nuclear force. Finally, ghost particle is a catch all term for particles with noisy measurements or particles that we do not fully understand.

The first classifier we wanted to build was one to determine if a particle sent through the accelerator was a ghost particle, based off of measurements inside of the LHC. To do this binary classification, we built classifiers using logistic regression, LDA, QDA, and decision trees, and tuned them using 10-fold cross validation. To test each model, we computed metrics such as total error, sensitivity, specificity, FNP, and FPP. The best model we were able to build was the decision tree, which had a total validation error of 9 percent.

After determining whether a particle is a ghost or not, physicists would next like to know what kind of particle they found. To solve this task, we built five different multi-class classification models and compared their performances to the validation data. The five models we looked at were Random Forests, Multinomial Logistic Regression, Naive Bayes, QDA, and LDA. The best performing model by far was the Random Forest, which had a final particle misclassification error rate of only 24 percent. All other models lagged behind, with LDA having the second best particle misclassification error rate of 27 percent. From this, we determined that CERN scientists could do well in predicting the class of particle that was detected in the collider using a random forest.

After building the two models, one for classification of ghost particles, and one for classifying particles into one of the 5 classes described above, we used the two models together to predict previously unseen test observations. The model ensemble we built had a test accuracy of 89 percent.

The rest of the report is organized as follows. First we did some exploratory data analysis to understand our high dimensional data. Next, we dive deeper into our analyses and explore some of the methods we were able to implement. Finally, we state our final findings and conclusions. Following the conclusions, we have an appendix with some more in depth explanation of the variables in our data set and some extra information for reference.

Exploratory Data Analysis

Since the LHC has so many detectors inside of it, physicists can generate all sorts of metrics that can be used as predictors. Due to this, our dataset has about 50 features. Because of this, we wanted to perform some dimensionality reduction to help visualize and understand the data. First, we wanted to perform PCA on the training data and then plot the first two principal components against each other to see if we could visualize any clusters in the data.

We plot the first two principal components and color the clusters based on the classes.

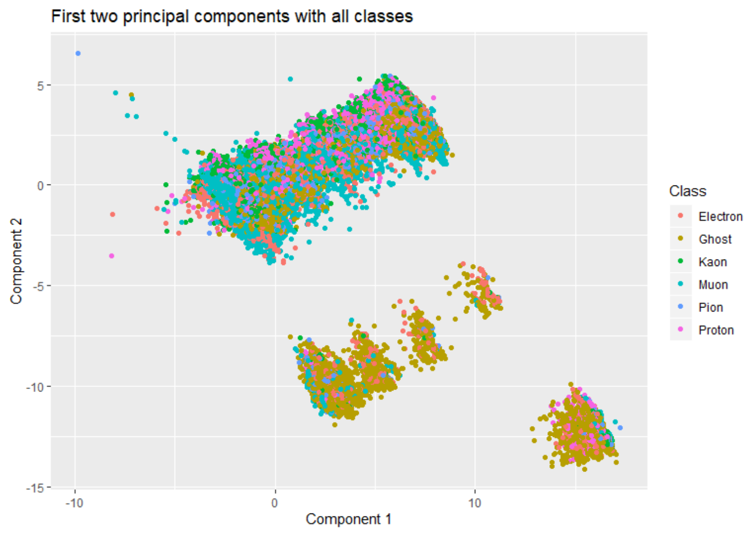

Figure 1: First two principal components of the data colored by their class association. There seem to be 6 distinct clusters. We see that ghost particles are found in many of the clusters. Other classes also seem to be dispersed throughout the different clusters and there does not seem to be any clear cluster-class association.

From this, it is evident that Ghost particles seem to be in every cluster. This follows from the fact that all noise is classified as Ghost. Therefore, performing multiclass prediction on this data would be difficult, since there are no particular physical constraints that define Ghost particles. Particularly, ghost particles are found in the clusters in the bottom right of the plot alongside protons and electrons, making it difficult to distinguish between them.

We now visualize the data with just Ghost and Not-Ghost classes. Now, the class-cluster distinctions are more evident.

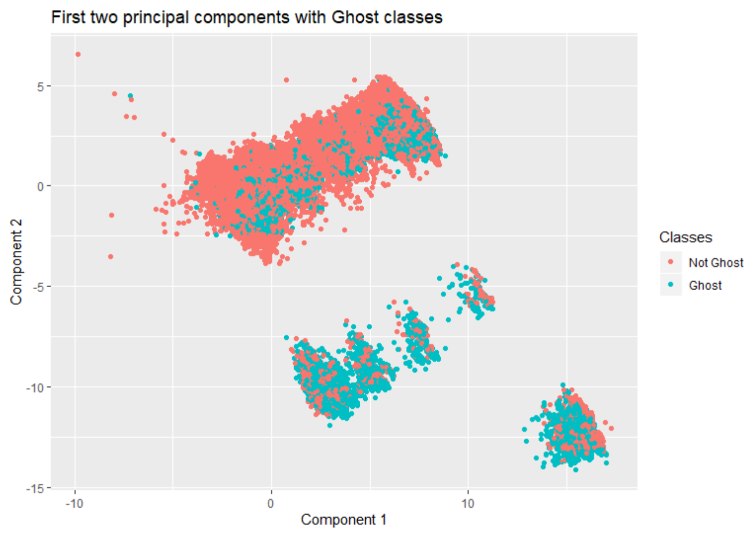

Figure 2: First two principal components of the data colored by their class association of Ghost or Not-Ghost. From this, we see clearer Ghost and Not-Ghost clusters. Here, we see that ghost particles are found mostly in the bottom right clusters, but the ghost particles in the top left cluster are also near each other.

Now we visualize the first two principal components with Ghost particles removed. From this visualization, we see more distinct associations between class and space on the plot.

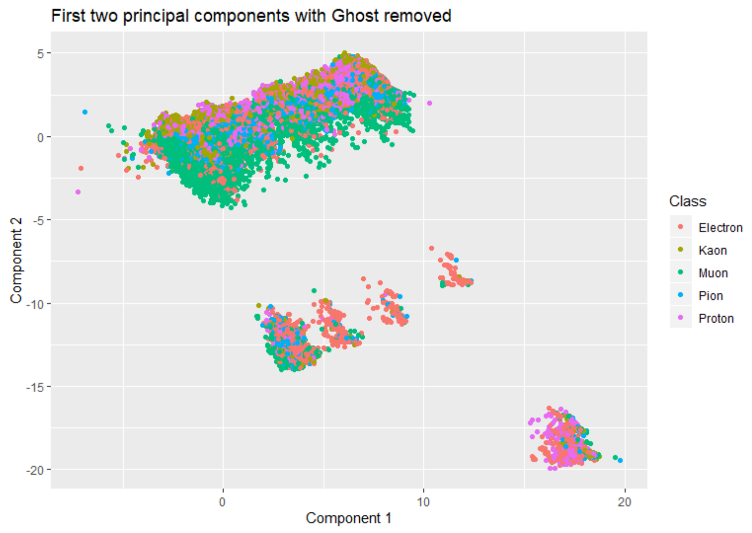

Figure 3: First two principal components of the data colored by their class association of Electron, Kaon, Muon, Pion or Proton. From this, we see that there are clearer clusters with specific particles occupying the majority of certain clusters. For example, with the removal of Ghost particles, the proton cluster in the bottom right corner of the plot becomes more evident.

From this exploration, it makes sense to break the analysis into two parts. First, we need to predict which particles are Ghost and Not-Ghost particles. Then, from the particles that are deemed to be Not-Ghost, we will identify the individual particle classes.

Analyses

For our analyses, we first build a model to determine whether a particle detected in the LHC was a ghost particle or not. Next, build a classifier to determine the class of particle that was detected, provided it was not a ghost particle. Putting these two models together, we show how well we can classify particles into the correct class.

Analysis 1:

For our first analysis, we wanted to train a model that could differentiate between Ghost particles and other particles. This analysis is pertinent for this investigation because Ghost particles have properties unlike other particles that could be attributed to noise or an unknown particle. Therefore, being able to differentiate between these particles and others is crucial in discovering new types of particles or for better classification of known particles. A model that could make such a classification based on the given tool would be important in reducing costs of LHC operations in particle identification and would assist greatly in the discovery of new particles.

Classification Pipeline:

1. We modified the labels into a binary classification problem by labeling Ghost particles with 1s and others as 0s.
2. The initial train data was further split into a train and validation set, with 80 percent of the data in the train set and 20 percent split into a validation set.
3. A logistic regression, LDA, QDA and decision tree model were trained on the new train data, with cross validation used for selecting hyperparameters.
4. The models were tested on the validation set. The model with the lowest error on the validation set was used for classification on the test data set.

Models:

(a) Logistic Regression:

For logistic regression, we fit the model on all of the parameters and performed cross validation to choose the best lambda value for regularization using the R-package glmnet. We used 10-fold cross validation to identify a best lambda value of 0.0003405875. Particularly, from **Figure 4** below, we can see that we choose the lambda value that is within 1 standard deviation of the value that minimizes the Binomial deviance, which is approximately -8 on the log λ scale. We choose the value within one standard error of the minimum lambda because generally this lambda performs better on validation data. This lambda is then used to tune the model and predict values on the validation set.

(b) LDA:

For LDA, we fit the model on all of the parameters using the R-package MASS and perform 10-fold cross-validation to choose the best tau-value by which to designate the decision boundary. That is, we selected the best value of tau (the threshold) by tuning the probability value at which we classify points as Ghost or Not-Ghost. From **Figure 5** below, we see that the tau value that minimized error was found to be $\tau = 0.35$. This model and decision boundary was then used to predict classes on the validation set.

(c) QDA:

For QDA, we fit the model on all of the parameters using the R-package MASS and perform 10-fold cross-validation to choose the best tau-value by which to designate the decision boundary. That is, we selected best value of tau by tuning to identify the probability value at which we classify points as Ghost or Not-Ghost. From the **Figure 6** below, we see that the tau value that minimized error was found to be $\tau = 0.95$. This model and decision boundary was then used to predict classes on the validation set.

(d) Decision Tree:

We trained a decision tree on all of the parameters and then performed 10-fold cross-validation to determine the best size of tree to return in the cost-complexity sequence using the R-package tree. That is, we used cross-validation to determine the best tree size to which to prune the larger decision tree by choosing which tree size had the smallest deviance, as displayed in the figure below. From **Figure 7** below, we see the size that minimized deviance is $k = 6$. Once the best size was determined, we pruned the decision tree and utilized the pruned tree to predict classes on the validation set. We used deviance as the measure of node heterogeneity used to guide cost-complexity pruning rather than misclassification rate since deviance generally does not have a tendency to overfit the model like misclassification may.

Hyperparameter Tuning Figures:

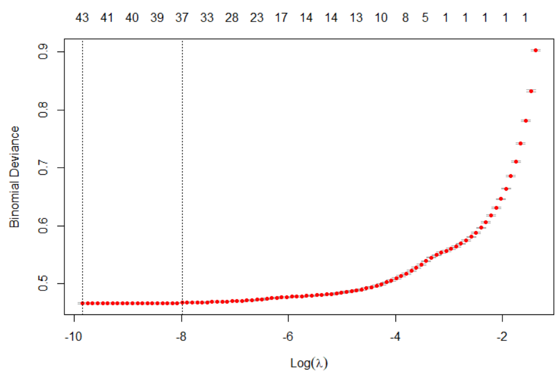

Figure 4: A plot of the Binomial Deviance vs Log(lambda) value for Logistic Regression. We want to choose the lambda that minimizes deviance as the hyperparameter for the model.

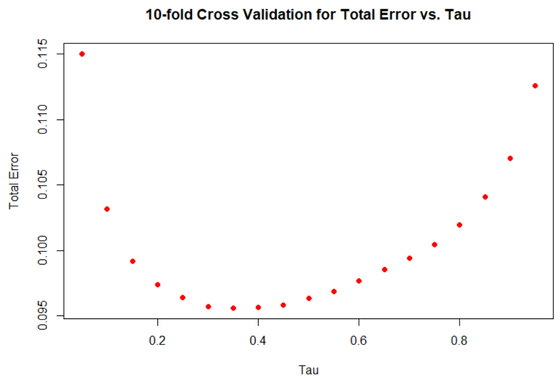

Figure 5: A plot of the Total Error vs Tau value for LDA. We want to choose the Tau that minimizes error as the hyperparameter for the model.

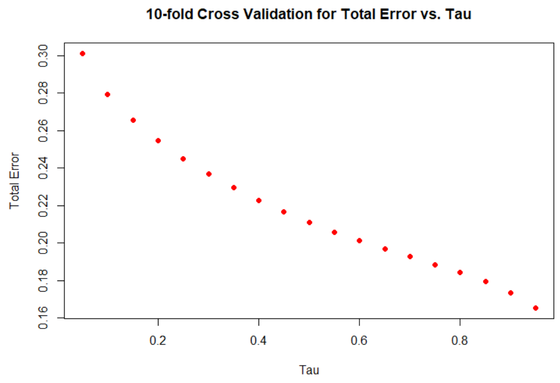

Figure 6: A plot of the Total Error vs Tau value for QDA. We want to choose the Tau that minimizes error as the hyperparameter for the model.

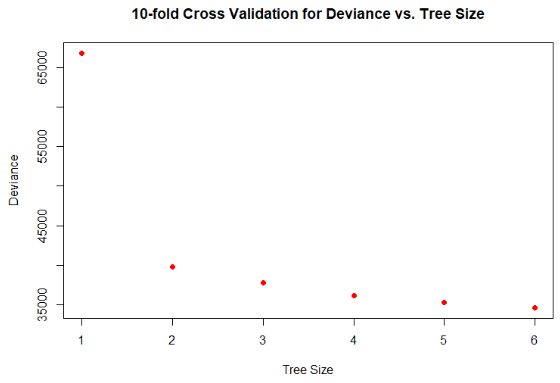

Figure 7: A plot of the Deviance vs Tree Size. We want to choose the Tree Size that minimizes deviance as the hyperparameter for the model.

Performance on Validation data:

From the table below, we see that logistic regression, LDA and Decision tree have errors that are within 0.003 of each other, indicating that they all perform similarly well on the validation data. QDA performs markedly worse, with nearly double the total error of 16 percent. The pruned decision tree model performs the best on this data with a total error of 0.0927. On the validation set, we see that generally, while there is a high specificity, i.e., values close to 1, for all of the models, the sensitivity of the models are much less impressive, around 0.6. This may be due to the nature of the data, which may contain irregularities that contribute to the high FPP and thus lower sensitivity across the models.

	Sensitivity	Specificity	Total Error	FNP	FPP
Logistic Regression	0.6042059	0.9677277	0.09335	0.07629355	0.2091665
LDA	0.6674933	0.9520022	0.0958	0.06588698	0.2625753
QDA	0.767037	0.8495963	0.164275	0.05246931	0.4926348
Decision Tree	0.6079258	0.9676576	0.09278333	0.0756358	0.2085109

Table 1: Describes the Sensitivity, Sensitivity, Total Error, FNP and FPP for every model. We can use these values to compare model performances.

Final Model Selection:

Despite three models performing similarly on the validation data, we choose the Decision Tree model to be the one we use to evaluate performance on the test set because it both has the lowest test error on the validation data and because it is the most interpretable model.

We believe to err on the side of interpretability because then we can understand what variables and tools are most important for classification. This way, not only will physicists have a model for classification, they will also know to watch out for outliers regarding specific variables in the data or pay attention to certain machines when performing experiments.

From the decision tree, it seems that the most important variables in determining the Ghost particles are GhostProbability, which was expected, but also TrackPt. The particle transverse momentum and the ghost probability as measured by the tooling are important for determination of the Ghost class.

Analysis 2:

After determining whether or not a particle sent through the accelerator was a ghost particle or not, the next step of the analysis is to determine the actual class of particle that was sent through the accelerator. For this task, there are 5 different kinds of particles that we can detect: Electrons, Protons, Kaons, Pions, and Muons. For physicists, knowing exactly which particle was found in the particle accelerator is particularly important to help understand the nature of the collision that took place inside the LHC. Unfortunately, it is difficult and expensive to determine the kind of particle that was sent through the collider, so machine learning can help reduce the cost and perhaps increase the accuracy of the results of the experiment. To accomplish this task, we wanted to build a number of multi-class classification models and test their accuracies against each other. The pipeline is shown below.

Multiclass Classification Pipeline:

1. After loading in the training set, we removed all observations of ghost particles.
2. The initial training data was further split into a train and validation set, with 80 percent of the data in the train set and 20 percent split into a validation set.
3. Random Forests, Naive Bayes, LDA, QDA and Logistic regression models were trained on the new train data.
4. The models were tested on the validation set. The model with the lowest error on the validation set was used for classification on the test data set.

Models:

The first statistical learning method we used was the random forest. Random forest runtimes are fast, and they work well with unbalanced data. This is particularly important for particle physics data, where certain types of particles are extremely rare or otherwise quite hard to detect. The problem with random forests is that they may overfit to noisy data. However, we have a 1.2 million row training data set, which may mitigate these issues. The random forest implementation we used came from the randomForest package in R. We set the parameter for the number of trees in the ensemble to 10, since the sheer size of our data made for very long run times. Next, we built 4 different models, each differing in the parameter "M" which is the number of variables randomly sampled as candidates for each split. For each of these models, we fit the model using the training data and determined accuracy using the validation set. The results for this hyperparameter tuning are shown in Table 2.

From the table, we can see the best value of M is 4, which had a validation error of 23.6 percent, so we select our final random forest to randomly sample 4 candidates for each split.

M - Value	Validation Error
2	0.2432716
3	0.2407995
4	0.2366659
5	0.2388478

Table 2: Random Forest Hyperparameter Tuning

The next model we wanted to implement was a multinomial Naive-Bayes classifier. These models are quick and easy to implement and the underlying architecture of the model is based on Bayes' Rule, which is easily interpretable for any scientifically literate person. The implementation we used to fit our Naive-Bayes model came from the e1071 package in R. The validation error rate was very high, almost 59 percent. Naive Bayes tends to work very poorly when predictors are not independent, and since many of the different predictors in our dataset are coming from measurements by the same few tools in the LHC, it makes sense that there may be some dependency structures in our data.

Next, we tried two different discriminant analysis models, QDA and LDA. The implementations for these models came from the MASS package in R. QDA validation error was not too much better than that of the Naive-Bayes model, with a value of 49 percent. LDA performed much better however, with a validation error of only 27 percent. From this, we can conclude that the decision boundary for this data set is more likely to be linear than nonlinear.

Finally, we also wanted to test the performance of multinomial logistic regression. The implementation we used for our model came from the R package nnet. On the validation set, logistic regression showed average performance, with a missclassification rate of 31 percent. The results of model performance are summarized in **Table 3**.

Model	Validation Error
Random Forest (M = 4)	0.2366659
Naive - Bayes	0.5876012
QDA	0.4884359
LDA	0.2749392
Logistic Regression	0.3062062

Table 3: Misclassification Error for Different Models

From table 3, we can see that the model with the lowest validation error is the Random Forest model. Next, we trained a Random Forest with $M = 4$ on the entire 1.2 million row training data. Looking at some of the model metrics, we learned a lot about the most important features used in determining the class of particle. These findings are summarized in **Figure 8** below. The delta log-likelihood values were most important in determining the class in particle, especially from the RICH detector. This makes sense, since the delta log-likelihoods are tools which estimate probability a particle falls into a certain class, depending on its interaction with the sensor inside the LHC. Unfortunately, these DLL values can be imprecise and contradictory, so they are imperfect predictors of particle type, but still very valuable.

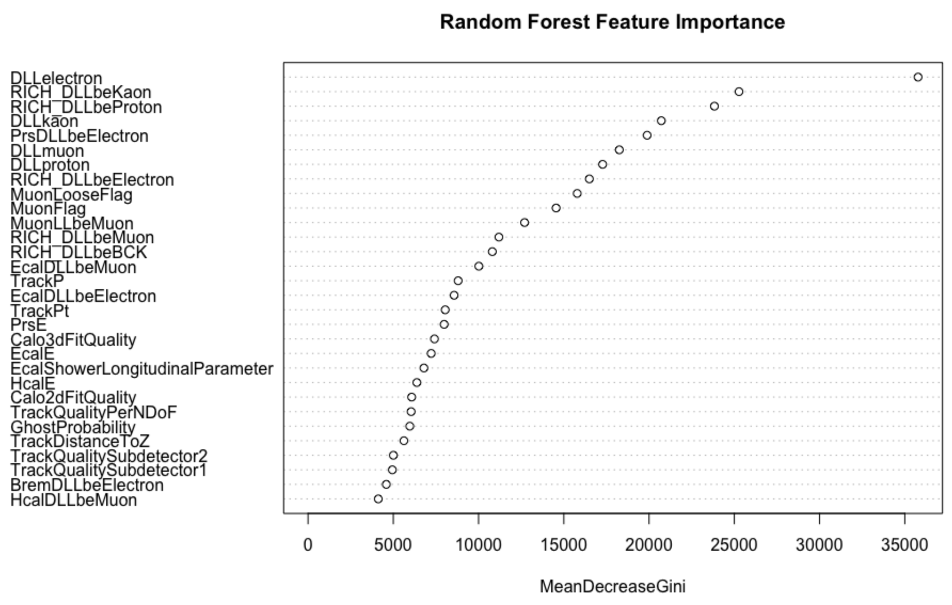

Figure 8: Graphical Representation of the feature importance for the Random Forest Model. The higher the mean decrease of the Gini metric, the higher the feature importance. From this, it's clear to see the most important features involve particle delta log-likelihoods.

Test Data Performance:

If physicists at CERN are want to determine what kind of particle was detected inside the LHC, they will need to make use of the models from both Analysis 1 and Analysis 2. Since true test data will not be labeled, the first thing they are going to have to do is determine which particles are Ghosts using the binary classification decision tree, and then use the multi-class random forest to label the non-Ghost observations as either Electrons, Protons, Kaons, Pions, and Muons. We modeled this pipeline using our test data, which consists of another 600000 observations. Luckily, we had labels and we were able to use these to test the accuracy of our modeling pipeline. The first thing we did was take the entire data set and apply the decision tree to determine which observations were Ghosts. Then, we applied the Random Forest to the remaining data to predict class. Using this pipeline, we were able to get to classify 89 percent of the test observations correctly. While these results are good for a first pass at the data, it is possible there may be more powerful methods to improve our results. Recent developments in deep learning via neural networks have been used in many problems in physics, and may be useful in next steps of this analysis.

Conclusions

From our analyses, we have found that we are able to create a pipeline between Analysis 1 and Analysis 2 that can predict the type of class with 89 percent accuracy. However, there are various points at which we could improve our process to generate results that may outperform this model.

In Analysis 1, we found that LDA, Logistic Regression and Decision Tree all produced models with similar, low, misclassification errors of about 9 percent. However, all of the models had Sensitivities around 60 percent, which is not ideal for this sort of model. This means that about 40 percent of data that are classified as Ghost particles are actually false positives. The total error is low because there are far fewer Ghost particles than Not-Ghost particles which artificially decreases the error rate due there not being equal amounts of Ghost and Not-Ghost items. However, all of the tested models performed poorly on this metric. This may be due to the fact that the Ghost class is assigned to noise that does not have clear behavior as determined by the instruments. Due to this, outlier values may be assigned to the Ghost class due to the size of our data. With more data, we would be better be able to differentiate between particle and noise within our data set. However, there is relatively high specificity across models, which means that many of the particles are being correctly classified as Not-Ghost, with few Ghost particles being identified as Not-Ghost. Therefore, it follows that due to the nature of our dataset, outlier Not-Ghost values are being picked up as noise by our classifiers and are being classified as Ghost values.

However, despite these issues, we have relatively low misclassification rates with the Decision Tree model as noted earlier. This decision tree utilizes the transverse momentum and Ghost probability measures to determine whether particles are Ghost or not. Other variables seem to be less important for this designation, as determined by the pruned decision tree model.

For the multiclass classification after the removal of Ghost particles, the Random Forest model with $M = 4$ performed better than the other models based on the validation set error. Naive Bayes and QDA performed particularly poorly, which may indicate that the independence assumptions for Naive Bayes do not hold and that the data are not separated by a non-linear decision boundary as expected by QDA. LDA and Logistic regression perform better, which may indicate that linear decision boundaries perform fairly well on this dataset. Random Forest worked best likely because it is flexible and because we have many data points. Since it is an ensembling algorithm, with more data points, we were able to combat overfitting and create an algorithm that took into account many of the subtle features of the data set.

Overall, using the decision tree for identifying Ghost particles and then the random forest for the multiclass classification of the rest of the particles, we only had 11 percent error. This includes the effect of the compounding error with misclassified Ghost particles when classifying the other types of particles. That is, if we had perfect knowledge of which particles were to be labeled as Ghost, we would have better performance for the classification of other classes.

Our results are somewhat satisfactory, as 89 percent accuracy is quite high. However, this accuracy is still not fantastic and obvious improvements could be made.

Particularly, for future improvements we could try more complex models such as neural networks. These models could identify more complex relationships between the variables. Even a simple feed-forward neural network may be able to determine better decision boundaries within our data set.

Furthermore, we could use regularization and perform more nuanced hyperparameter tuning on our models to combat overfitting. Also, instead of fitting on all of the predictors, we could attempt some sort of variable selection and fit on the important variables. This may improve some of the models.

In all, while we performed a comprehensive analysis, there are more powerful, computationally-intensive and time-consuming methods that we could use to provide more accurate models.

Appendix

In this section, we go into depth about the variables included in our analyses. Abbreviations are as follows: Spd stands for Scintillating Pad Detector, Prs - Preshower, Ecal - electromagnetic calorimeter, Hcal - hadronic calorimeter, Brem denotes traces of the particles that were deflected by detector.

1 ID - Identification value for tracks (presents only in the test file for the submitting purposes). We removed this variable when performing statistical analyses.

2 Label - string valued observable denoting particle types. Can take values ”Electron”, ”Muon”, ”Kaon”, ”Proton”, ”Pion” and ”Ghost”.

3 FlagSpd - flag (0 or 1), if the particle track passes through Scintillating Pad Detector

4 FlagPrs - flag (0 or 1), if reconstructed track passes through Preshower detector

5 FlagBrem - flag (0 or 1), if reconstructed track passes through a deflector

6 FlagEcal - flag (0 or 1), if reconstructed track passes through Ecal

7 FlagHcal - flag (0 or 1), if reconstructed track passes through Hcal

8 FlagRICH1 - flag (0 or 1), if reconstructed track passes through the first RICH detector

9 FlagRICH2 - flag (0 or 1), if reconstructed track passes through the second RICH detector

10 FlagMuon - flag (0 or 1), if reconstructed track passes through muon stations (Muon)

11 SpdE - energy deposit associated to the track in the Spd

12 PrsE - energy deposit associated to the track in the Prs

13 EcalE - energy deposit associated to the track in the Hcal

14 HcalE - energy deposit associated to the track in the Hcal

15 PrsDLLbeElectron - delta log-likelihood for a particle candidate to be electron using information from Prs

16 BremDLLbeElectron - delta log-likelihood for a particle candidate to be electron using information from Brem

17 TrackP - particle momentum

18 TrackPt - particle transverse momentum

19 TrackNDoFSubdetector1 - number of degrees of freedom for track fit using hits in the tracking sub-detector1

20 TrackQualitySubdetector1 - chi2 quality of the track fit using hits in the tracking sub-detector1
21 TrackNDoFSubdetector2 - number of degrees of freedom for track fit using hits in the tracking sub-detector2
22 TrackQualitySubdetector2 - chi2 quality of the track fit using hits in the tracking sub-detector2
23 TrackNDoF - number of degrees of freedom for track fit using hits in all tracking sub-detectors
24 TrackQualityPerNDoF - chi2 quality of the track fit per degree of freedom
25 TrackDistanceToZ - distance between track and z-axis (beam axis)
26 Calo2dFitQuality - quality of the 2d fit of the clusters in the calorimeter
27 Calo3dFitQuality - quality of the 3d fit in the calorimeter with assumption that particle was electron
28 EcalDLLbeElectron - delta log-likelihood for a particle candidate to be electron using information from Ecal
29 EcalDLLbeMuon - delta log-likelihood for a particle candidate to be muon using information from Ecal
30 EcalShowerLongitudinalParameter - longitudinal parameter of Ecal shower
31 HcalDLLbeElectron - delta log-likelihood for a particle candidate to be electron using information from Hcal
32 HcalDLLbeMuon - delta log-likelihood for a particle candidate to be using information from Hcal
33 RICHpFlagElectron - flag (0 or 1) if momentum is greater than threshold for electrons to produce Cherenkov light
34 RICHpFlagProton - flag (0 or 1) if momentum is greater than threshold for protons to produce Cherenkov light
35 RICHpFlagPion - flag (0 or 1) if momentum is greater than threshold for pions to produce Cherenkov light
36 RICHpFlagKaon - flag (0 or 1) if momentum is greater than threshold for kaons to produce Cherenkov light
37 RICHpFlagMuon - flag (0 or 1) if momentum is greater than threshold for muons to produce Cherenkov light
38 RICH_DLLbeBCK - delta log-likelihood for a particle candidate to be background using information from RICH
39 RICH_DLLbeKaon - delta log-likelihood for a particle candidate to be kaon using information from RICH
40 RICH_DLLbeElectron - delta log-likelihood for a particle candidate to be electron using information from RICH
41 RICH_DLLbeMuon - delta log-likelihood for a particle candidate to be muon using information from RICH
42 RICH_DLLbeProton - delta log-likelihood for a particle candidate to be proton using information from RICH
43 MuonFlag - muon flag (is this track muon) which is determined from muon stations
44 MuonLooseFlag muon flag (is this track muon) which is determined from muon stations using looser criteria
45 MuonLLbeBCK - log-likelihood for a particle candidate to be not muon using information from muon stations
46 MuonLLbeMuon - log-likelihood for a particle candidate to be muon using information from muon stations
47 DLLelectron - delta log-likelihood for a particle candidate to be electron using information from all subdetectors
48 DLLmuon - delta log-likelihood for a particle candidate to be muon using information from all subdetectors
49 DLLkaon - delta log-likelihood for a particle candidate to be kaon using information from all subdetectors
50 DLLproton - delta log-likelihood for a particle candidate to be proton using information from all subdetectors
51 GhostProbability - probability for a particle candidate to be ghost track. This variable is an output of classification model used in the tracking algorithm.

Citations and References

1. Collins et. al. ”Extending the Bump Hunt with Machine Learning” https://arxiv.org/abs/1902.02634

2. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. New York: Springer, 2013.

3. Guest et. al, ”Deep Learning andits Application to LHC Physics” https://arxiv.org/abs/1806.11484