Udacity Machine Learning Engineering Nanodegree
Capstone Project
Gary Foreman

I. Definition

Project Overview:

Gamma rays (the highest energy forms of light) give astronomers insight into some of the most extreme events in our universe. Given their high energies, gamma rays are not produced as part of normal thermal processes, e.g., stellar fusion, because the temperatures required would be much too high. Rather, gamma rays are produced in non-thermal processes such as black hole accretion and shockwaves created by stellar explosions (also known as supernovae).

Gamma rays lie well beyond the range of light that is visible to our human eyes, and their detection requires substantial efforts of science and engineering. Gamma-ray telescopes rely upon measurements of interaction products created when gamma rays pass through another medium. For ground-based telescopes such as the one providing data for this capstone, that medium is Earth's atmosphere, whereas for space-based telescopes, heavy metals are used to induce electron-positron pair creation (beyond the scope of this capstone).

(Side note: this project of interest to me because I used measurements from a gamma-ray telescope to carry out my PhD research. I have been removed from the field for a few years, but this project will be a nice way to reconnect.)

The detection of gamma rays requires the filtering of noise from other high energy, charged, particles, usually protons and electrons (also known as cosmic rays). Cosmic rays produce similar, but not exactly the same, characteristics within a gamma ray detector; however, cosmic rays originate from different sources, and confuse the signal from gamma ray sources. This cosmic-ray confusion makes gamma ray sources appear brighter than they truly are if the cosmic-ray signal is not properly removed.

The Major Atmospheric Gamma Imaging Cherenkov (MAGIC) Telescopes form a ground-based array used for measuring gamma-ray light. Ultimately, they detect interaction products from gamma rays and particles in Earth's atmosphere. As mentioned above, cosmic rays produce similar interaction products, but their signal must be removed before accurate gamma ray measurements can be made. Here, we build a product to distinguish gamma rays from cosmic rays based on data from MAGIC.

Problem Statement:

We leverage the publicly available MAGIC Gamma Telescope Data Set to build a machine learning model to classify events as either gamma rays or cosmic rays (aka hadrons). This data is provided by the University of California Irvine Machine Learning Repository. Original work with this data set dates back to Bock et al. (2004), when scientists were experimenting with machine learning models to use as part of MAGIC's data preprocessing pipeline. We seek to outperform the models presented by Bock et al. (2004) given 15 years of advancements in both compute hardware and machine learning theory. The authors provide benchmarks to beat in their receiver operating characteristic curve, which they present in their Figure 5.

In a realistic situation in which a gamma-ray telescope would be providing data to the public, a convenient interface to a model which distinguishes signal (gamma rays) from noise (hadrons) must be provided to allow scientists to study celestial sources using accurate data. AWS provides not only a good solution for hosting our machine learning model itself, but also provides the means for us to host a RESTful API. This API would allow for a telescope observing event data to distinguish in real time whether an event is a gamma ray or a hadron. This way, signal can already be separated from noise before the data is presented to scientists for further analysis. As discussed below, our solution includes interface to our model through the Amazon API Gateway, a Lambda function to parse input in JSON format to a vector consumable by our model, and a model endpoint responsible for making the gamma ray/hadron prediction.

Metrics:

As mentioned above, Bock et al. (2004) present the performance of the models the explored in a ROC curve. While they do not explicitly report the AUC of their models, we will optimize our models using AUC. In addition, the description of the data on the University of California Irvine site highlights several import false positive rate thresholds at which the true positive rate should be optimized. These false positive rate thresholds 0.01, 0.02, 0.05, 0.1, and 0.2, and we will report performance at these thresholds as another means of comparison with the Bock et al. (2004) models.

II. Analysis

Data Exploration:

As documented on the University of California Irvine site, there are 19,020 rows in this data set; 12,332 of which are gamma-ray events and 6,688 of which are cosmic-ray events. The target is imbalanced with 65% of records being gamma rays. All feature columns are continuous. Ten features and one target column are included in the data provided at the UCI hosting site; below, find the column descriptions reproduced from that page:

1. fLength: continuous # major axis of ellipse [mm]
2. fWidth: continuous # minor axis of ellipse [mm]
3. fSize: continuous # 10-log of sum of content of all pixels [in #phot]
4. fConc: continuous # ratio of sum of two highest pixels over fSize [ratio]
5. fConc1: continuous # ratio of highest pixel over fSize [ratio]
6. fAsym: continuous # distance from highest pixel to center, projected onto major axis [mm]
7. fM3Long: continuous # 3rd root of third moment along major axis [mm]
8. fM3Trans: continuous # 3rd root of third moment along minor axis [mm]
9. fAlpha: continuous # angle of major axis with vector to origin [deg]
10. fDist: continuous # distance from origin to center of ellipse [mm]

11. class: g,h # gamma (signal), hadron (background)

This data contains no missing values and therefore no imputation is required. In Table 1, we present a sample of the data provided at the UCI website. In Table 2, we provide summary statistics of the feature set.

	Length	Width	Size	Conc	Conc1	Asym	M3Long	M3Trans	Alpha	Dist	class
0	24.6014	12.8664	2.3793	0.5637	0.2985	17.3748	19.5657	8.2758	22.8210	216.9520	g
1	49.9223	22.3316	3.2930	0.2363	0.1210	6.7223	51.4013	-12.9949	6.8500	240.1190	g
2	28.1635	15.8070	2.4200	0.4259	0.2490	-17.9803	-13.7842	-8.3405	17.1252	248.0300	g
3	138.3880	44.5241	3.2860	0.0949	0.0674	-71.2351	66.0265	-48.3917	45.0480	250.2907	h
4	80.7882	55.0349	3.4713	0.1449	0.0725	31.8274	70.9064	58.1480	22.5290	277.3980	g

Table 1: Sample training data.

	Length	Width	Size	Conc	Conc1	Asym	M3Long	M3Trans	Alpha	Dist
count	15216.00	15216.00	15216.00	15216.00	15216.00	15216.00	15216.00	15216.000	15216.00	15216.00
mean	53.15821	22.14692	2.825405	0.379758	0.214244	-4.05386	10.75720	0.215030	27.61418	194.0297
std	42.08143	18.31808	0.470501	0.182416	0.110233	58.82876	50.82770	20.834744	26.03460	74.44742
min	4.283500	0.000000	1.941300	0.013100	0.000300	-457.916	-331.780	-205.8947	0.000000	1.282600
25%	24.33557	11.89855	2.478375	0.235100	0.128000	-20.4943	-12.7313	-10.86555	5.577325	142.9692
50%	37.26630	17.14955	2.740250	0.353100	0.195950	4.001900	15.33220	0.446100	17.70900	192.3025
75%	70.18922	24.69205	3.103775	0.504100	0.285100	23.89770	35.96497	10.901050	45.78142	240.7069
max	334.1770	256.3820	5.323300	0.893000	0.674000	575.2407	238.3210	179.85100	90.00000	495.5610

Table 2: Feature summary statistics.

Exploratory Visualization:

In Figure 1, we plot feature distributions compared against each other and their correlation with the target variable. The plot below does indicate a degree of correlation between some of our features, most notably Conc and Conc1. Correlation will not necessarily be a concern for the models built here, but would be a problem if we were interested in investigating feature importance. The univariate distributions plotted on the diagonal do not indicate any particularly strong correlation between individual features and the target. This suggests a well-performing model will likely need to leverage multiple features simultaneously to make predictions.

In Figure 2, we compute and visualize the Pearson correlation coefficients of our feature variables. As seen in the previous plot, the Conc and Conc1 features are highly correlated. Because we are not specifically interested in determining feature

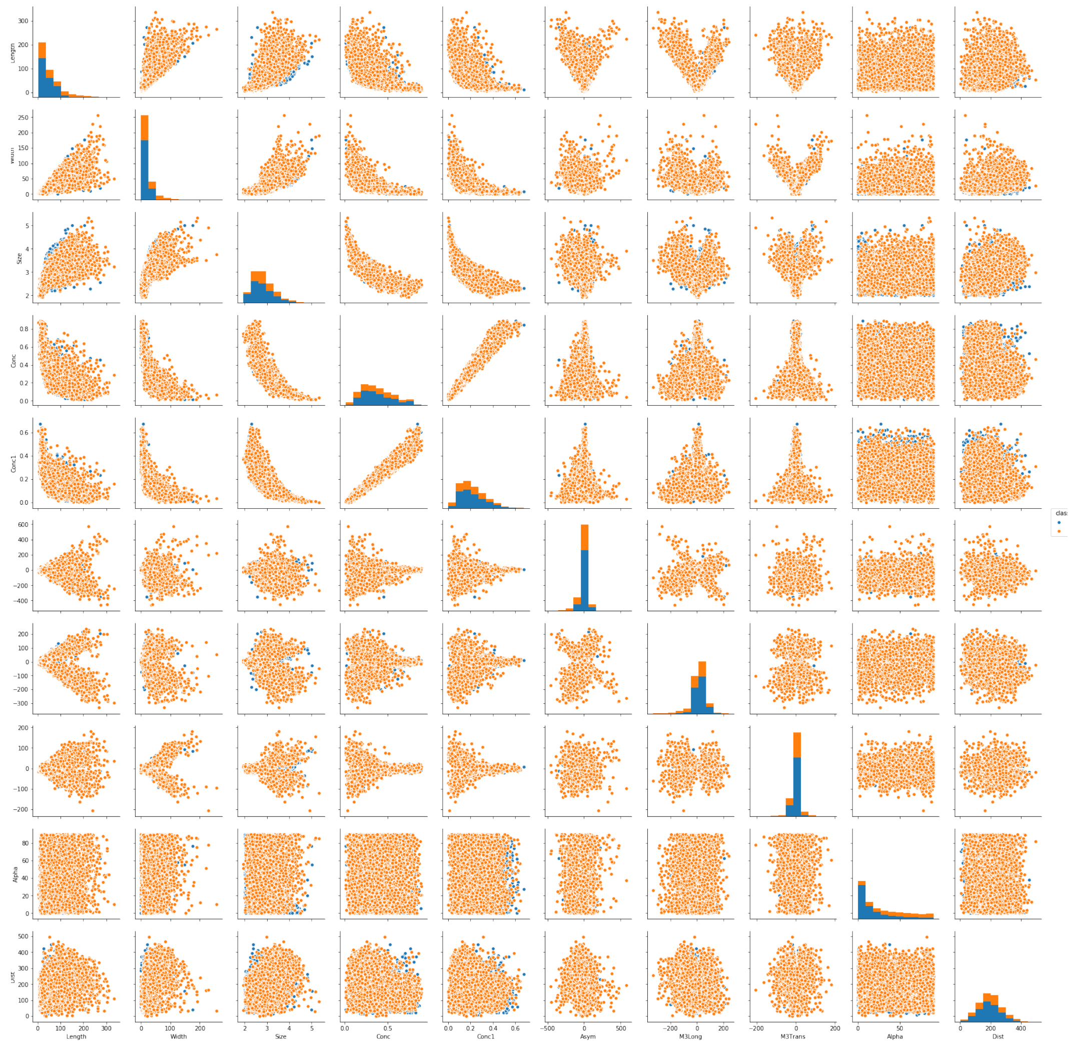


Figure 1: Pairwise feature plot (please see the eda/eda.ipynb notebook for code used to generate this plot).

importance here, we note the correlation, but keep both features in our training data set.

Algorithms and Techniques:

For our deployed production model, we train a gradient boosting classifier using AWS's xgboost implementation. This algorithm includes several hyperparameters, which we optimize using cross validation of our training set. Performance of our final model will be reported using a randomly selected test set that is 20% of the data provided by the UCI host site. We will use this test set to compare results to those presented by Bock et al. (2004) as well as to our own benchmark models described below.

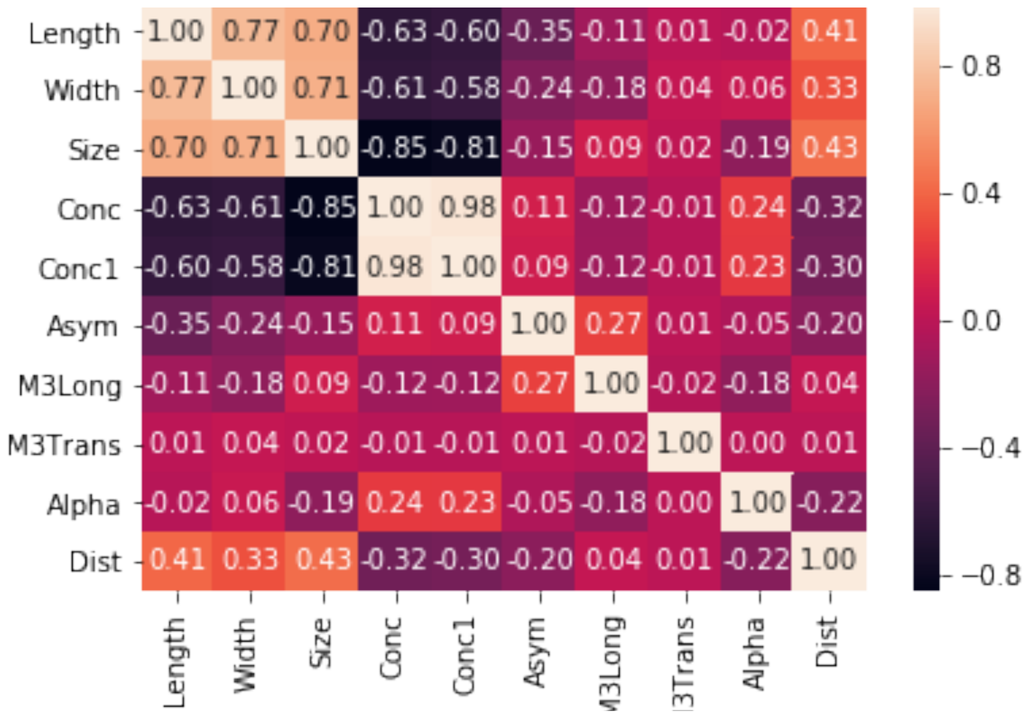


Figure 2: Pearson's correlation coefficient (please see the eda/eda.ipynb notebook for code used to generate this plot).

A gradient boosting classifier is well suited to this problem due to its ability to tease out non-linear relationships between our feature sets. As mentioned in the Exploratory Visualization section, individual features do not tend to distinguish the target class very well, so these interactions will be necessary for a well-performing model. In addition, the gradient boosting classifier will require no data preprocessing in our case where features are continuous and no values are missing. However, gradient boosting classifiers aren't without their drawbacks. Training for this algorithm is relatively expensive compared to, say, the logistic regression algorithm; there are more hyperparameters to optimize, and the RAM required to store the decision trees that form the foundation of the algorithm can be quite large. Also, gradient boosting classifiers are more costly at time of prediction than logistic regression. As opposed to simply plugging features into a mathematical formula, a gradient boosting model must traverse an array of binary decision trees before a prediction can be made. Fortunately, given the relative rarity of detected gamma-ray events (especially compared with photon events in the optical wavelengths), the lag during predict time should not be a concern.

Benchmark:

As discussed above, we use the results presented in Bock et al. (2004) as our benchmark. Those authors ensembled various algorithms to achieve their peak reported performance. Unfortunately, Bock et al. (2004) do not provide numerical values for the performance of their models, but only visualize them in the ROC curve they present in Figure 5. In addition to the Bock results, we train our own benchmark models to serve as a source of a quantitative comparison. We use both logistic regression and k-nearest neighbors algorithms to provide relatively simple yet predictive watermarks for our gradient boosting classifier to beat. We leverage a test set randomly generated from the data provided by UCI as a way to compare our benchmark models to the gradient boosting classifier.

III. Methodology

Data Preprocessing:

For the k-nearest neighbors and logistic regression algorithms used as our benchmarks, features must be scaled so that they are distributed over approximately the same range of values. We can see from Table 1 above that the Conc feature, for example, lies in a range of about 0 to 1, whereas the Dist feature lies in a range from 0 to 500. Left unadjusted, these types of discrepancies will lead to poor performance of the k-nearest neighbors and logistic regression algorithms. However, this is not an issue for the tree-based gradient boosting classifier algorithm.

For the gradient boosting classifier trained and deployed using AWS SageMaker, we needed to reformat our training and testing data sets for compliance with S3 and the requirements of SageMaker's implementation of the XGBoost algorithm. This includes uploading csv files for both the train and test sets. These files must contain no headers or row indices, and the train set must have the target as its first column.

Implementation:

The code to develop and deploy the production model to AWS can be found in the producton_model/production_model.ipynb notebook in the GitHub repository that accompanies this document.

We port optimized hyperparameters described in the next section for our gradient boosting classifier model to an xgboost model trained using the Amazon SageMaker interface. As a side note, we chose not to perform exploratory data analysis and initial model training/hyperparameter optimization using SageMaker because throughout this course, we found SageMaker to inhibit the iterative model build process. The overhead associated with working in the cloud, especially for a data set of this size, didn't makes sense and development was faster on our laptop.

We leverage an image based on the XGBoost algorithm for the deployment of our production model. As detailed in the next section, we leverage hyperparameters optimized during local training for our deployed model so that we do not need to use SageMaker for the iterative portion of our model build process. We set max_depth=5, eta=0.1 (eta corresponds to learning_rate), and num_rounds=300 (num_rounds corresponds to num_estimators); we leave all other parameters set to default. While the Scikit-Learn and XGBoost implementations are based on the same underlying Gradient Boosting algorithm, their exact implementations differ in the details. We were encouraged to find consistent performance of both implementations.

Next we set up the infrastructure for an API that can be used to send data to and receive predictions from our deployed model. The first necessary decision was how we expect input data to be formatted. We have chosen a JSON format as follows:

‘{“Length”:21.0483,“Width”:17.6112,“Size”:2.4409,“Conc”:0.4529,“Conc1”: 0.2373,“Asym”:29.591,“M3Long”:-9.6241,“M3Trans”:14.2623,“Alpha”:32.4415,“Dist”: 173.549}’

with the feature name as the key and the feature itself as the value. We then wrote a method that parses this JSON string to produce a csv string of simply the feature values ordered as expected by the model. We run through the prediction steps for a single event from our test set (the one listed above) as a means to determine the necessary code to include in our Lambda function.

Finally, we include the code we implement in our Lambda function at the end of production_model/production_model.ipynb. This function is responsible for parsing the input JSON string mentioned above, transforming that string to a format consumable by our model endpoint, calling the model, and returning the result to the API gateway for display to the end user. Our Lambda function returns a “g” for gamma ray (signal) if the model prediction probability is above 0.5, otherwise it returns “h” for hadron (noise).

Our model API is hosted at https://baf900kck2.execute-api.us-east-2.amazonaws.com/prod, and is intended to be called using curl with a POST command. Figure 3 shows two example API calls including the response. The first call is an example from our test set that is labeled “g,” and the second is an example from our test set labeled “h.” We see in both cases that the model prediction is correct. We will leave the model, Lambda function, and API deployed until a response is received from our reviewer. This way they reviewer may test the API if they choose.

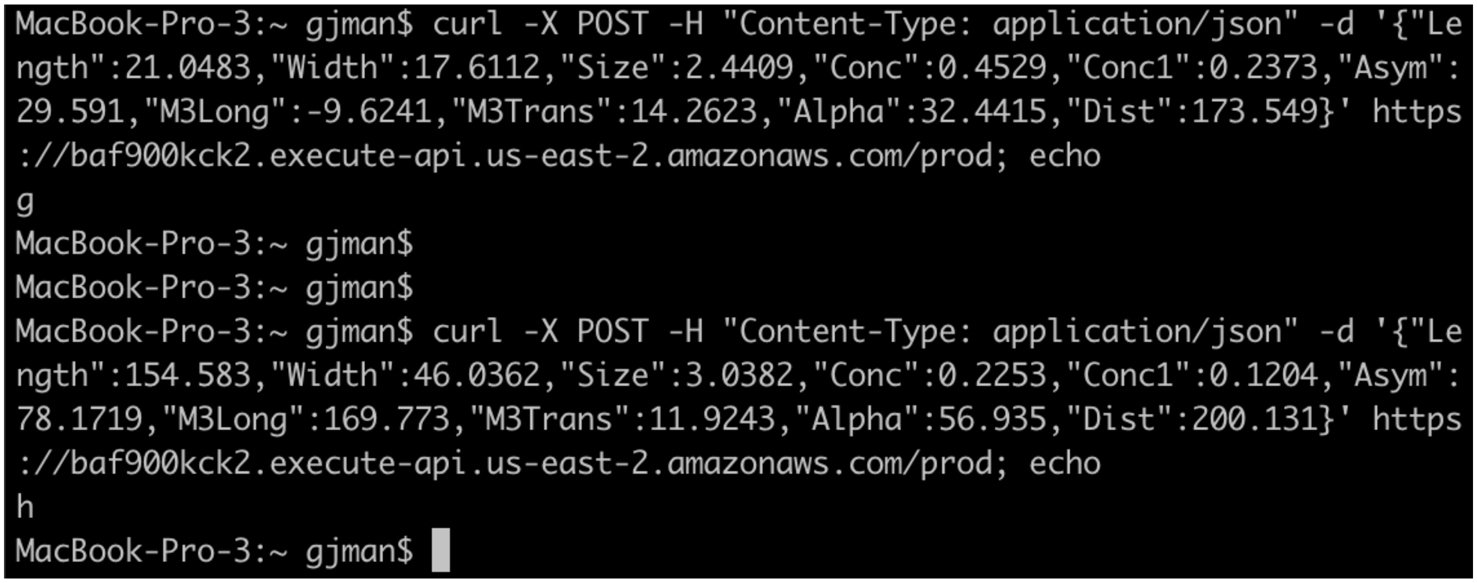
Refinement:

The model refinement reported here is performed in the eda/eda.ipynb notebook that can be found in the GitHub repository that accompanies this report.

As indicated above, we performed hyperparameter optimization outside of SageMaker. SageMaker does not implement cross validation as a means of hyperparameter optimization, and given the relatively small size of our data set, we find this to be a prohibitive shortcoming. We train a gradient boosting classifier. Using 4-fold cross validation, we optimize the n_estimators, learning_rate and max_depth hyperparameters. An optimal solution with mean test AUC of 0.93 is achieved using a learning_rate of 0.1, a max_depth of 5, and with 300 n_estimators.

As a benchmark, we train a k-nearest neighbors classifier. We optimize the n_neighbors hyperparameter using 4-fold cross validation. The results of the grid search indicate that the k-nearest neighbors algorithm performs surprisingly well. The optimal mean test AUC of just under 0.9 across validation folds was achieved using 37 neighbors.

We also train a logistic regression classifier. We optimize the L2 regularization strength hyperparameter C using 4-fold cross validation. Logistic regression performs substantially worse than k-nearest neighbors, which may be due to what we discovered in the pairwise plots above: individual features do not distinguish the target very well. We found performance to be independent of regularization strength probably because our model includes so few features. The logistic model consistently performs at just under 0.84 AUC across validation folds.



```
MacBook-Pro-3:~ gjman$ curl -X POST -H "Content-Type: application/json" -d '{"Le
ngth":21.0483,"Width":17.6112,"Size":2.4409,"Conc":0.4529,"Conc1":0.2373,"Asym":
29.591,"M3Long":-9.6241,"M3Trans":14.2623,"Alpha":32.4415,"Dist":173.549}' https
://baf900kck2.execute-api.us-east-2.amazonaws.com/prod; echo
g
MacBook-Pro-3:~ gjman$
MacBook-Pro-3:~ gjman$
MacBook-Pro-3:~ gjman$ curl -X POST -H "Content-Type: application/json" -d '{"Le
ngth":154.583,"Width":46.0362,"Size":3.0382,"Conc":0.2253,"Conc1":0.1204,"Asym":
78.1719,"M3Long":169.773,"M3Trans":11.9243,"Alpha":56.935,"Dist":200.131}' https
://baf900kck2.execute-api.us-east-2.amazonaws.com/prod; echo
h
MacBook-Pro-3:~ gjman$
```

Figure 3: POST requests to the API hosting the XGBoost model trained on the MAGIC-gamma data. The first request is an example from the test set that is labeled “g,” and the second request is a test set record labeled “h.” Our model predicts both cases correctly. Note: the echo command is included purely so that a new line character is created after the API response; this is for readability purposes only.

IV. Results

Model Evaluation and Validation:

We test performance of the three models presented above using a test set created by holding out a random 20% of the original data. We find AUC of 0.84, 0.90, and 0.94 for the logistic regression, k-nearest neighbors, and gradient boosting classifier, respectively; these results are consistent with the AUCs reported while performing cross validation during hyperparameter optimization.

In Tables 3 through 5, we present the the true positive rates for each of our three models (two benchmarks and production) at the critical false positive rates (1%, 2%, 5%, 10%, and 20%) highlighted in the Attribute Information section of the MAGIC Gamma Telescope Data Set site hosted by the University of California, Irvine. We can see from Table 5 that the gradient boosting classifier outperforms out benchmark models and the best results from Bock et al. (2004) at each of the key thresholds.

	fpr	tpr
20	0.009673	0.031707
48	0.020089	0.109756
122	0.049851	0.293902
232	0.099702	0.493089
447	0.200149	0.726829

Table 3: ROC for logistic regression model.

	fpr	tpr
1	0.005952	0.190650
2	0.017113	0.326423
4	0.044643	0.523171
6	0.093750	0.662602
10	0.195685	0.819512

Table 4: ROC for k-nearest neighbors model.

	fpr	tpr
26	0.009673	0.268699
52	0.020089	0.449187
128	0.049851	0.641870
238	0.099702	0.780081
426	0.199405	0.921951

Table 5: ROC for gradient boosting classifier.

Justification:

As we can see from Tables 3 through 5, the gradient boosting classifier consistently and often substantially outperforms our benchmark models at key points on the ROC curve. From the Bock et al. (2004) ROC curve presented in their Figure 5, at a 0.01 false positive rate, their best performing model has a true positive rate of around 0.27, consistent with our production model. At 0.02, their best reported model is well below 0.4 whereas ours is nearly 0.45. At 0.05, their best reported model is right around 0.6 as opposed to our 0.64. At 0.1, they achieve a true positive rate of just over 0.7, whereas our gradient boosting classifier is at 0.78. And at a false positive rate of 0.2, their best model has a true positive rate of less than 0.9, and our gradient boosting classifier has a true positive rate of 0.92.

V. Conclusion

Free-form Visualization:

We use our randomly selected test set to create Figure 4, the the receiver operating characteristic (ROC) curve for the three models trained here. This plot is meant to be directly compared with Figure 5 of Bock et al. (2004). Black vertical lines are places at 1%, 2%, 5%, 10%, and 20% false positive rate to highlight key thresholds outlined under the Attribute Information section of the MAGIC Gamma Telescope Data Set site hosted by the University of California, Irvine. These results are presented in Tables 1 through 3. Our gradient boosting model outperforms all methods presented by Bock et al. (2004) at each of the required false positive thresholds.

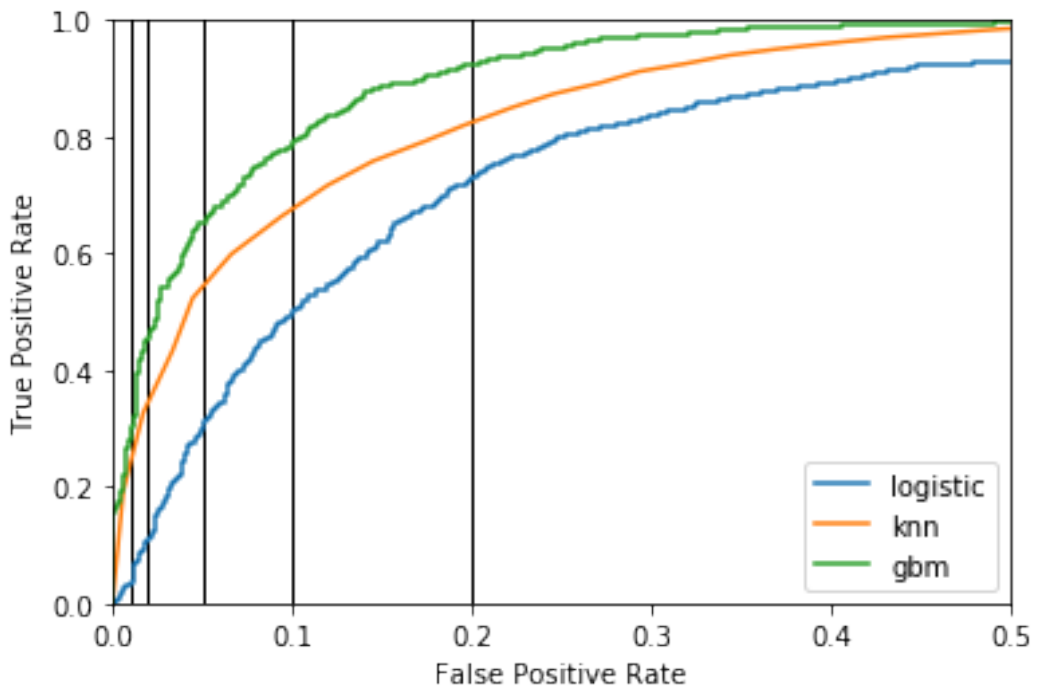


Figure 4: Receiver operating characteristic. Black vertical lines indicate thresholds identified by the data contributors. (Please see the eda/eda.ipynb notebook for code used to generate this plot.)

Reflection:

We have trained and deployed a gradient boosting classifier to distinguish gamma-ray from cosmic-ray (i.e. hadron) events. While our model was optimized locally using Scikit-Learn's implementation of this algorithm, we found optimal hyperparameters translated will to SageMaker's XGBoost implementation. In both cases, the AUC on our test set was 0.94. This is well above anything that was achieved by Bock et al. (2004), and is also comfortably above our two benchmark models, logistic regression(AUC of 0.84) and k-nearest neighbors (AUC of 0.9). We have also written and deployed an API interface for our model using an AWS Lambda function and the AWS API Gateway. The API is hosted at https://baf900kck2.execute-api.us-east-2.amazonaws.com/prod and can be called using the curl unix function.

Improvement:

One opportunity for future work lies in further sophistication of the implemented machine learning algorithm. Artificial neural networks could also provide a good framework for modeling this data. As we mentioned in our Exploratory Visualization section, no individual feature clearly distinguishes the gamma ray and hadron classes, and well-performing machine learning models must be able to capture interactions between features. There is really no rival to the neural network algorithm when it comes to automatically detecting and leveraging feature interactions. However, this benefit comes with the added cost of high risk of overfitting, long training times, and a wide array of hyperparameters to optimize. Given more time to complete this capstone, my focus would have moved to neural networks next.

Another opportunity to improve the algorithm would be ensembling of various model types. As we have mentioned throughout this report, the UCI data description highlights key points on the ROC curve at which performance should be validated. There is nothing to say that a single machine learning algorithm needs to perform best at all of these points on the curve. One algorithm could be optimal at say a false positive rate at 0.01, and another algorithm could perform better at 0.2. In our case, the gradient boosting classifier outperformed our benchmark models at all points, this might not be the case if we threw an artificial neural network into the mix.

The final improvement that could be made to the API is to allow the caller to pass their required false positive rate threshold. If the user were to request a false positive rate of 0.02, for example, we would only predict gamma ray if our output model probability was 0.98 or above. On the other hand if the user requested a false positive rate of 0.2, we would predict gamma at a probability threshold of 0.2. This would not only require further sophistication from our Lambda function, but it would require a well calibrated model, which we have not focused on in this report.