Capstone Project

Machine Learning Engineer Nanodegree

Harry Shaun Lippy
April 22nd, 2016

Definition

Project Overview

The field of astronomy is rife with huge amounts of data that often require a substantial amount of an astronomer's time to sort through in order to extract meaningful information. Oftentimes telescopes are trained on certain parts of the sky and are "turned on" in order to record information for extended periods of time.

One such example of this data collection is found in the subfield of gamma ray astronomy. In this branch of astronomy, specialized telescopes are used to collect gamma rays that are emitted as the result of events such as supernovae and hypernovae, as well as by pulsars.[1] There are a number of active projects in the field, and one of the telescopes that has been developed in order to collect and analyze gamma rays is called the MAGIC telescope.

The MAGIC telescope (see Figure 1) is actually a combination of two imaging atmospheric Chernekov telescopes, which collect electromagnetic radiation for analysis. The telescope is located near the top of the "Roque de los Muchachos" on the island of La Palma in the Canary chain of islands.[2]

In the collection of data from the MAGIC telescope, there is a great deal of background noise that accompanies the desired signals. This background information must be identified and separated out from the signal in order to perform the analysis. Separating the signal from the noise can be a tedious process that consumes a large amount of the valuable time of the astronomers, and it would be of great benefit if a computer could do a lot of the preliminary "legwork" in order to reduce the workload on the astronomers. Machine learning is the best candidate process for doing this preliminary sorting of the data, and this project will experiment with a number of machine learning algorithms in order to provide a tool for astronomers to use to help with the analysis of signals received from the MAGIC telescope.

A data set of simulated events from a MAGIC gamma ray telescope is available for download from the University of California at Irvine Machine Learning Repository[3]. This data is simulated using Monte Carlo methods to provide a realistic repository from which a learning algorithm can be trained and tested.

[1] https://en.wikipedia.org/wiki/Gamma-ray_astronomy
[2] https://magic.mpp.mpg.de/home/
[3] https://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope

Figure 1 - MAGIC gamma ray telescopes (source: https://magic.mpp.mpg.de/newcomers/introduction/)

Problem Statement

The data set provides a large collection of events which can be used to train a classifier to distinguish between background noise (which comes from Hadronic showers initiated by cosmic rays in the atmosphere[4], and which is referred to as a "Hadron" event in the data set) and signal (events caused by primary Gamma sources, and referred to as a "Gamma" event in the data set). A small number of features is provided for each data point, and the goal will be to use these features, along with the class (Hadron or Gamma), to perform supervised learning to train a classifier that can then be used to sort through new data in order to pick out Gamma events for further analysis by an astronomer.

Since this is a problem of supervised learning – and more specifically one of binomial classification – methods of classification will be used in order to attack the problem. Several algorithms will be explored, as discussed below in the Analysis phase.

Metrics

In the description of the data set provided, the provider has very clearly specified that the use of a simple classification accuracy will not be sufficient for this problem. The reason given is that classifying a background event as a signal event is to be considered worse than classifying a signal as a background event.[5]

With this rubric in mind, the metric used to evaluate the performance of a given classifier will be the area under the ROC curve. An ROC curve is one that plots the false positive rate against

[4] https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.names [4. Relevant Information]

[5] https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.names [9. Class Distribution]

the true positive rate for a given data set using a given classifier. A larger area results in a more successful classifier. A graph plotting the ROC curves for various machine learning classifiers will be given so that the best algorithm can be selected visually by inspecting the graphs.

Analysis

Data Exploration

There are ten features given for this data set, and one column indicating the correct classification for the given data point (to be used as the target in supervised learning)[6]. A description of each feature, along with its data type, is given in Table 1.

Feature	Type of Data	Description
fLength	Continuous	Major axis of ellipse (mm)
fWidth	Continuous	Minor axis of ellipse (mm)
fSize	Continuous	10-log of sum content of all pixels (#phot)
fConc	Continuous	Ratio of sum of two highest pixels over fSize (ratio)
fConc1	Continuous	Ratio of highest pixel over fSize (ratio)
fAsym	Continuous	Distance from highest pixel to center, projected onto major axis (mm)
fM3Long	Continuous	Third root of third moment along major axis (mm)
fM3Trans	Continuous	Third root of third moment along minor axis (mm)
fAlpha	Continuous	Angle of major axis with vector to origin (degrees)
fDist	Continuous	Distance from origin to center of ellipse (mm)
Class	g, h	The target. g = gamma, h = hadron

Table 1 – Description of features of the data set

Two example data points from the data set, one each for a classification of g (gamma) and h (hadron), can be seen in Table 2.

fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	Class
31.5125	19.2867	2.9578	0.2975	0.1515	38.1833	21.6729	-12.0726	17.5809	171.227	g
93.7035	37.9432	3.1454	0.168	0.1011	53.2566	89.0566	11.8175	14.1224	231.9028	h

Table 2 – Example data points

[6] https://archive.ics.uci.edu/ml/machine-learning-databases/magic/magic04.names [7. Attribute Information]

The data set contains 19,020 data points, each of which is fully defined. A statistical analysis of the data has been performed, with results given in Table 3.

FEATURE	MIN	MAX	MEAN	MEDIAN	STD
fLength	4.28	334.18	53.25	37.15	42.36
fWidth	0.00	256.38	22.18	17.14	18.35
fSize	1.94	5.32	2.83	2.74	0.47
fConc	0.01	0.89	0.38	0.35	0.18
fConc1	0.00	0.68	0.21	0.20	0.11
fAsym	-457.92	575.24	-4.33	4.01	59.21
fM3Long	-331.78	238.32	10.55	15.31	51.00
fM3Trans	-205.89	179.85	0.25	0.67	20.83
fAlpha	0.00	90.00	27.65	17.68	26.10
fDist	1.28	495.56	193.82	191.85	74.73

Table 3 – Basic statistical analysis of the data set

As can be seen from the basic statistical analysis, there are several features – fSize, fConc, and fConc1 – which are of a different scale than the other features. For this reason, it is decided that normalizing and centering the data (with mean 0 and variance 1) will be beneficial. This will be done in the pre-processing stage of the learning algorithm.

In addition to the basic statistical analysis, an outlier analysis was done on the data set. Each data point was checked to see if four or more of the feature columns for a given data point lie outside of three standard deviations from the mean for that column. Any data point that meets this threshold is considered an overall outlier data point. There were only 161 such points.

There are no missing values and no categorical columns for the data set.

Exploratory Visualization

A scatter matrix plot can be used to determine whether or not there are any interesting, especially linear, relationships between the various features in the data set. In addition, it is possible to use the matrix to determine the density of each of the features (i.e., is it Gaussian?)

Figure 2 contains a scatter plot matrix obtained from the data set for this problem. In order to prevent the plot data from being unreadable due to the high number of data points, it was decided to randomly sample (with replacement) 10% of the data from the data set for the analysis. As can be seem from the plots, the only apparent pairwise linear relationship occurs between fConc and fConc1, which is expected since these two features are measuring nearly the same thing (fConc is a ratio of the sum of the two highest pixels over the fSize, whereas fConc1 is simply the ratio of the one highest pixel over the fSize). Also, by looking on the diagonal in the plot, which represents the estimation of the density for each of the features, we can see that each of the features appears to be normally distributed. Several features – fLength, fWidth, fSize and fAlpha - are strongly positively skewed. fConc and fConc1 also appear to be positively skewed, although less strongly. fAsym, fM3Long, and fDist are not skewed positively or negatively. The final feature, fM3Trans, appears to be bimodal.

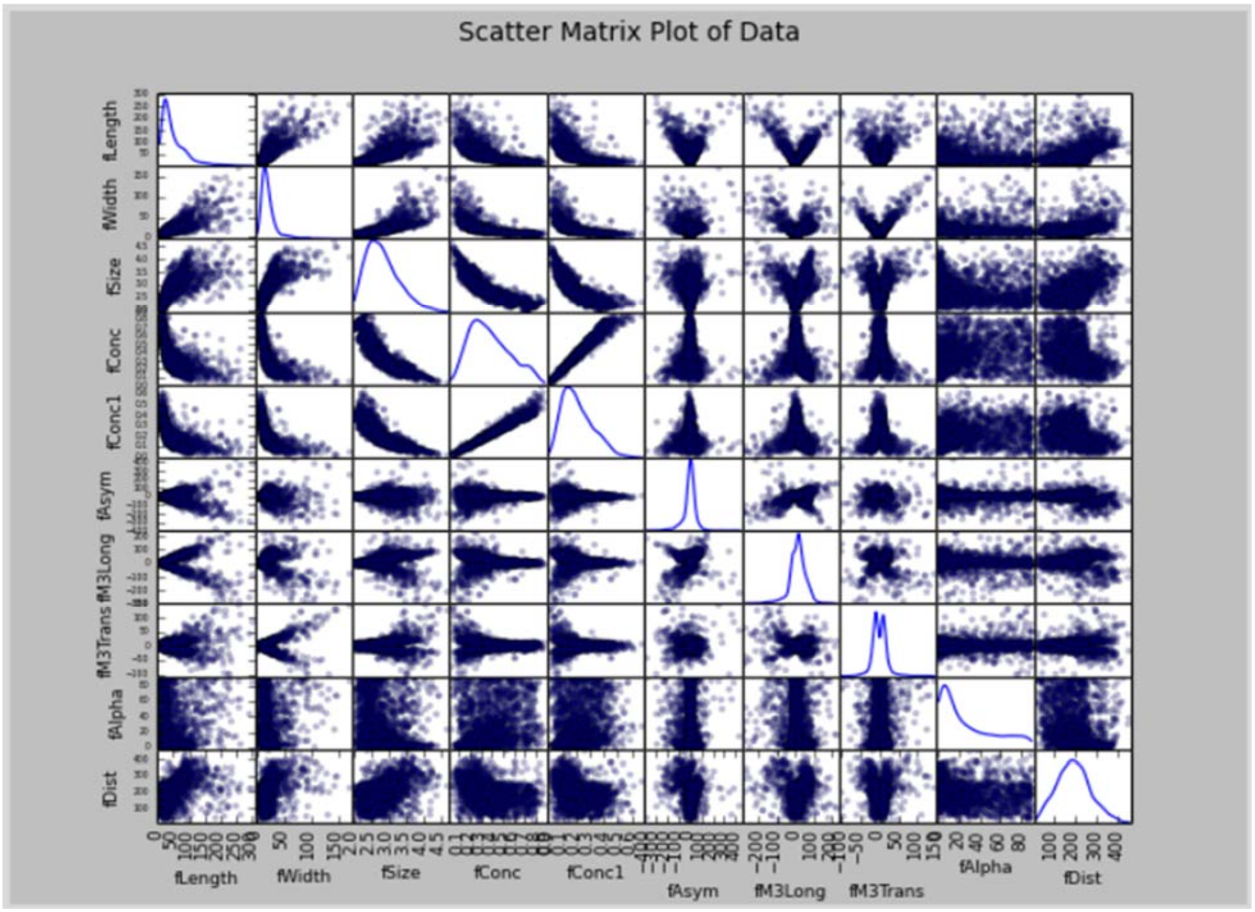

Figure 2 – Scatter Matrix Plot of the Data (10% random sample)

Algorithms and Techniques

Given that the problem is one of supervised learning, and more specifically one of binomial classification, there are a great deal of algorithms available for training a classifier to learn from this data. Algorithms were selected taking into account the nature of the problem, as well as taking into account the analysis on the data set described above. We have a fairly large number of data point (more than19,000) and a small number of features (10), so this will inform our choice of algorithms.[7]

It was decided to try three algorithms of differing complexity, each of which will be described below. For each algorithm, a number of parameters were selected which will be searched using the grid search cross validation technique in order to select the best value for that parameter (see “Implementation” in the “Methodology” section for more details on the process of parameter searching). Each algorithm has a large number of other default parameters which can be examined by navigating to the URL within the sklearn

[7] http://scikit-learn.org/stable/tutorial/machine_learning_map/

documentation for the given algorithm. (URLs will be given as footnotes for each of the classifiers chosen.)

1. **Random Forest**
    a. *Description:* A Random Forest is an ensemble method that uses what is known as "feature bagging" to create a series of decision trees where, at each step, a random sample (with replacement) of the data is drawn in addition to a random subset of the features available. So each decision tree will be trained on a different subset of the features. The method increases bias, but this increase in bias is usually accompanied by a decrease in variance that is more than enough to compensate.[8] It seems like this algorithm will be useful for this data set, as it is not clear from the visual inspection which subset of features would be best for making decisions via a decision tree classifier; by basing decisions on an average of a number of trees created from random subsets of features, the model could be better.
    b. *sklearn class:* RandomForestClassifier[9]
    c. *Default parameters selected for search:*
        i. n_estimators – the number of trees in the forest
        ii. max_depth – the maximum depth of a tree
        iii. criterion – the function used to measure the quality of a split. Values can be 'gini' for the Gini Impurity[10] or 'entropy' for the information gain[11]
2. **Support Vector Machine.**
    a. *Description:* A Support Vector Machine (SVM) works by attempting to find a separating hyperplane with the largest possible gap between the data points. A complex mathematical procedure dubbed "the kernel trick" allows this method to be used for data sets that have a very high dimensional space (i.e. have a large number of features). It is a very popular method for use in classification problems in supervised learning. The method is somewhat slower than the other methods used for this problem (see "Results" for timings), but the benefit is improved performance.
    b. *sklearn class:* SVC[12]
    c. *Default parameters selected for search:*
        i. kernel – the type of kernel to be used by the SVM. Values can be 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed', or a custom kernel designed by the user. Due to the long training times of this problem, only 'rbf' (the default) and 'poly' kernels were attempted (see "Improvement" in the "Conclusion" section for why we might want to try other kernels).
        ii. degree – degree of the polynomial for 'poly' kernels.
        iii. C – penalty parameter of the error term.
3. **K-Nearest Neighbors.**
    a. *Description:* K-Nearest Neighbors (K-NN) attempts to classify a given data point based on the classifications of nearby points, using a weighting system so that the "closest" neighbors have the most influence on the classification. "Closest"

[8] http://scikit-learn.org/stable/modules/ensemble.html#forest [1.11.2.1 Random Forests]
[9] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
[10] https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity
[11] https://en.wikipedia.org/wiki/Decision_tree_learning#Information_gain
[12] http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

can be defined as Euclidean distance, Minkowski distance[13], or others. For this problem, Minkowski distance, the default metric, was used as it was not clear how any other metric would result in a better classifier for this problem. An advantage of K-NN over the other algorithms used here is that it is quite easy to understand and its results tend to be trusted since the method can be grasped readily, with very little mathematical background.

b. *sklearn class:* KNeighborsClassifier[14]
c. *Default parameters selected for search:*
    i. n_neighbors – number of neighbors to use in the computation
    ii. weights – weight function to be used in prediction. Possible values are ‘uniform’, ‘distance’, or custom.
    iii. algorithm – algorithm used to compute the nearest neighbors. Possible values are ‘ball_tree’[15], ‘kd_tree’[16], ‘brute’, or ‘auto’.
    iv. leaf_size – leaf size used by the ‘ball_tree’ and ‘kd_tree’ algorithms

Benchmark

No readily accessible results for the given data set are available, and as such it is not possible to set an experimentally justified benchmark for the data set. As such, it is necessary to use other means by which to set a benchmark to determine whether or not the algorithms used for this problem are successful.

The University of Nebraska Medical Center has an interesting article discussing the use of the ROC curve for interpreting the usefulness of a diagnostic test in the field of medicine. In their estimation, an effective guide for interpreting an roc_auc_score in order to establish the success of a given test at correctly differentiating between disease and non-disease is the “traditional academic point system”[17]

- .90-1 = excellent (A)
- .80-.90 = good (B)
- .70-.80 = fair (C)
- .60-.70 = poor (D)
- .50-.60 = fail (F)

Of course, getting a diagnosis wrong in a medical classification procedure is going to be far more problematic than getting a diagnosis wrong in classifying an astronomical event. However, it seems reasonable to go by a similar point system in determining the success of an algorithm for the MAGIC gamma telescope data set, perhaps adjusted slightly down. With this in mind, a roc_auc_score of .85 or higher will be considered an excellent result for the algorithm, while a score of .80 to .85 will be considered a good result.

[13] https://en.wikipedia.org/wiki/Minkowski_distance
[14] http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
[15] https://en.wikipedia.org/wiki/Ball_tree
[16] https://en.wikipedia.org/wiki/K-d_tree
[17] http://gim.unmc.edu/dxtests/ROC3.htm

Methodology

Data Preprocessing

There was no missing data included in the given data set, so no preprocessing work was required to fill in missing values. The solution code does, however, have a line included to use interpolation to fill in missing values. This was added to build robustness into the solution so that adding data points to the data set will not require additional coding work should some of the new data be missing values.

Since the scale of values differs amongst the features, a step was included to center and normalize the data – that is, each column was scaled so that the resulting column zero mean and unit variance.

Finally, as was noted in the "Data Exploration" section, there were 161 outliers which were described in that section as data points where four or more of the columns for that data point lie more than three standard deviations from the mean for that column. Those data points were removed from the data set.

Implementation

The steps required to implement the steps needed to solve the problem are all contained within a Python class called MLNPCapstone. Within this class are contained methods (highlighted in blue in this section) which carry out the steps needed to load and prepare the data, perform the statistical analysis, preprocess the data, train the classifiers, and output the results.

The first step in the implementation of the learning algorithms was to split the data set into a training set and a testing set using the sklearn function train_test_split[18] from the package sklearn.cross_validation. 80% of the data was reserved for training the classifier, while 20% was set aside for testing the classifier once it has been trained. This split was chosen due to the abundance of data in the data set – if the data set had been smaller, reserving 20% for testing might not have been feasible. Note that this split happens after the preprocessing of the data has been completed, and is performed in split_dataframe.

Once the data set had been split up into training and testing sets, the classifiers could be built and trained in the train_classifier method. Several parameters for each of the classifiers were fed to the sklearn GridSearchCV[19] function from the sklearn.grid_search. The grid search is performed using K-fold cross-validation (k = 3) performed on a classifier trained using each possible permutation of the input parameters. As mentioned in the "Metrics" section under "Definition" above, the primary metric used to score the classifiers is area under the ROC curve. To implement this, the roc_auc_score[20] metric was passed to the GridSearchCV function as the scorer.

[18] http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.train_test_split.html
[19] http://scikit-learn.org/stable/modules/generated/sklearn.grid_search.GridSearchCV.html
[20] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_auc_score.html

The parameters for which optimal values were searched in each of the algorithms are described in Table 4 (for details on the meaning of the parameters, see the “Algorithms and Techniques” topic in the “Analysis” section).

CLASSIFIER	PARAMETER	POSSIBLE VALUES
RandomForestClassifier	n_estimators	[6,7,8,9,10,11,12,13,14,15]
	max_depth	[1,2,3,4,5,6,7,8,9,10]
	criterion	[‘gini’, ‘entropy’]
SVC	kernel	[‘rbf’, ‘poly’]
	degree (‘poly’ kernel only)	[2,3,4,5]
	C	[1.0, 10.0, 100.0, 1000.0]
KNearestClassifier	n_neighbors	[5, 10, 15, 20]
	algorithm	[‘ball_tree’, ‘kd_tree’, ‘auto’]
	weights	[‘uniform’, ‘distance’]
	leaf_size	[10, 20, 30]

Table 4 – Parameters to be searched for each learning algorithm

Once the GridSearchCV function returns from its search, the classifier is then fit on the training data using the fit method on the classifier.

Finally, once the classifier has been trained and fit, a score can be computed for the classifier using the score method on the classifier. This is done in the run method.

The primary complication that occurred during the implementation phase was related to the searching of the parameter space using GridSearchCV. It is very difficult to perform a very thorough search of the possible parameter space (which can admittedly be very large, particularly for integer or real-valued parameters) in a reasonable amount of time. The small subset of parameters and values listed in Table 4 were settled on because they seemed to have the biggest effect on the score from the limited subset that I tried. Due to the exhaustive nature of the search, adding just a couple of parameters and associated values increased the time for the search to complete exponentially. The SVC classifier in particular was slowed dramatically by adding just a few more values or even one more parameter. Obviously, restricting the number of parameters and values searched has the consequence of possibly missing the optimal combination.

Refinement

The Random Forest Tree classifier was selected for further refinement (see “Results” section for the reasoning behind this choice). The refinement step will be to simply dig down deeper into the parameter space in order to fine-tune the parameters to produce the best possible classifier.

Table 5 shows the additional parameters searched for when doing the additional fine-tuning. As expected, the training time when doing the refinement takes a considerably longer amount of time (again, see Table 6 in the “Results” section). However, as will be seen below, the fine-tuning does result in a slightly better classifier, so the extra time is profitably spent.

PARAMETER	POSSIBLE VALUES
max_features	[1,2,3,4,5,6,7,8,9,10]
min_samples_split	[4,5,6]
min_samples_leaf	[1,2,3]
oob_score	[True, False]
warm_start	[True, False]

Table 5 – Additional parameters added to search space for refinement of Random Forest Classifier

Results

Model Evaluation and Validation

Training and testing times for each of the classifiers attempted is given in Table 6. Clearly, the Support Vector Machine is by far the slowest of the classifiers in terms of both training and testing times. The training time for the K-Nearest Neighbors classifier is the smallest, while that of the Random Forest Classifier is much less than the Support Vector Machine. The testing time for the RFC is by far the lowest.

CLASSIFIER	TRAINING TIME (s)	TESTING TIME (s)
RandomForrestClassifier	120.495693	0.007569
RandomForestClassifier (refined)	598.489069	0.008019
SVC	434.284598	0.762522
KNearestClassifier	75.903594	0.384121

Table 6 – Training and testing times for the various classifiers

Table 7 shows the results of the classifiers when tested on the reserved test data (a visualization of the results can be seen in the ROC curve given in Figure 3, in “Free-Form Visualization” in the “Conclusion” section). It is evident that the RFC and SVM classifiers perform better than K-NN. In this and nearly every run that was undertaken, the RFC slightly outperformed the SVM. So the final model chosen for the problem is the RFC.

For the refined Random Forest Classifier, a learning curve has been plotted which shows the accuracy of the classifier as a function of the training set size. This plot is given in Figure 3. We see that cross-validation score improves, albeit slightly, with the increasing number of samples. The training score appears to level off at around 6500 samples at 0.89.

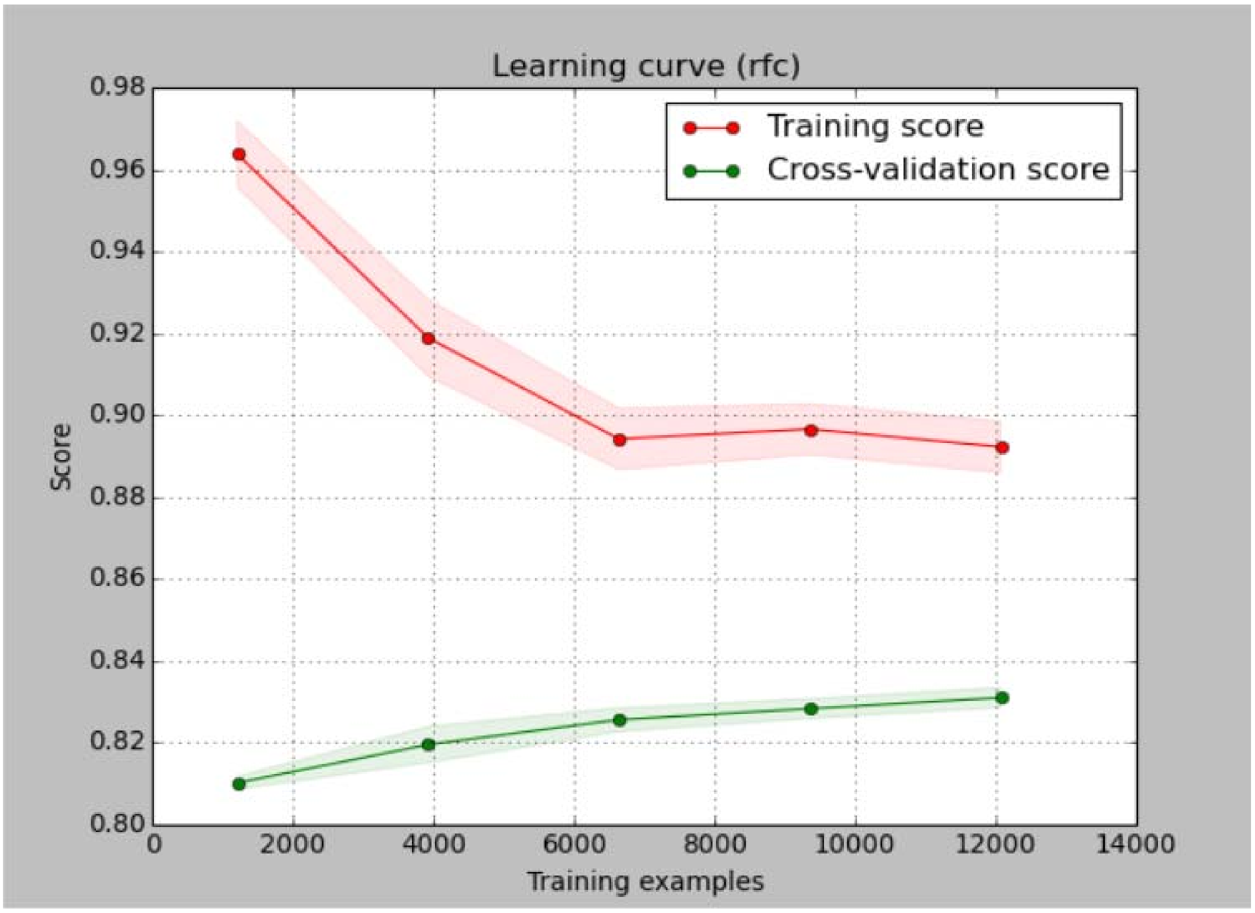

Figure 3 – Learning Curve for refined Random Forest Classifier

Justification

While the solution didn't produce a classifier that could be considered 'excellent' by the benchmark standard given above, it did produce a 'good' classifier which should be very beneficial to a scientist looking for help in classifying MAGIC telescope data. Table 8 shows the results of a cross-validation run of the final model selected (the Random Forest Classifier) on r using the K-Fold cross-validation procedure with k = 3, and it can be seen that the results are stable and close in value to the 'excellent' benchmarking number of 0.85. It should be noted that k = 3 was chosen due to the size of the data set; larger values of k resulted in cross-validation runs that took far too long to complete.

It is clear, from Table 7, that the K-Nearest Neighbors model was not sufficient to solve this problem – it seems highly unlikely that even with further refinement this model would perform to the 'excellent' level set by the benchmark. The Support Vector Machine model, however, could possibly be made 'excellent' with further refinement, although the extremely long training times for this algorithm (see Table 6) make further refinement extraordinarily difficult.

CLASSIFIER	PARAMETERS	auc_roc_score
RandomForrestClassifier	criterion='gini', max_depth=10, n_estimators=13	0.831950
RandomForestClassifier (refined)	criterion='gini', max_depth=10, n_estimators=13, min_samples_split=4, bootstrap=False, max_features=4, min_samples_leaf=2, warm_start=True	0.835795
SVC	degree=2, kernel='rbf', C=1000.0	0.821314
KNearestClassifier	algorithm='ball_tree', leaf_size=10, n_neighbors=5, weights='distance'	0.792648

Table 7 – roc_auc_score for the various classifiers when run on the test data

.

FOLD	auc_roc_score
1	0.825782
2	0.822858
3	0.830925

Table 8 – Results of a K-fold (k=3) cross-validation run of the refined Random Forest Classifier model

Conclusion

Free-Form Visualization

An ROC curve was produced that compared the various algorithms used to develop a solution to the problem. As was discussed above in the "Definition" section (under "Metrics"), the primary metric used to determine the success of the models was the area under this ROC curve. Having a visual representation of the curve helps to determine, at a glance, which model is best. The ROC curve resulting from one run of the program is shown in Figure 4.

Reflection

The project began with a statistical analysis of the data set, from which it was determined that centering and normalization of the data was appropriate. In addition, a small number of outliers were detected in which four or more of the feature columns for the outlier data points had outlier values for that column. It was decided to remove these outliers from the data set before training and testing. Once the preprocessing was done, the data set was split into training and test sets, with 20% of the data set aside for testing the various models.

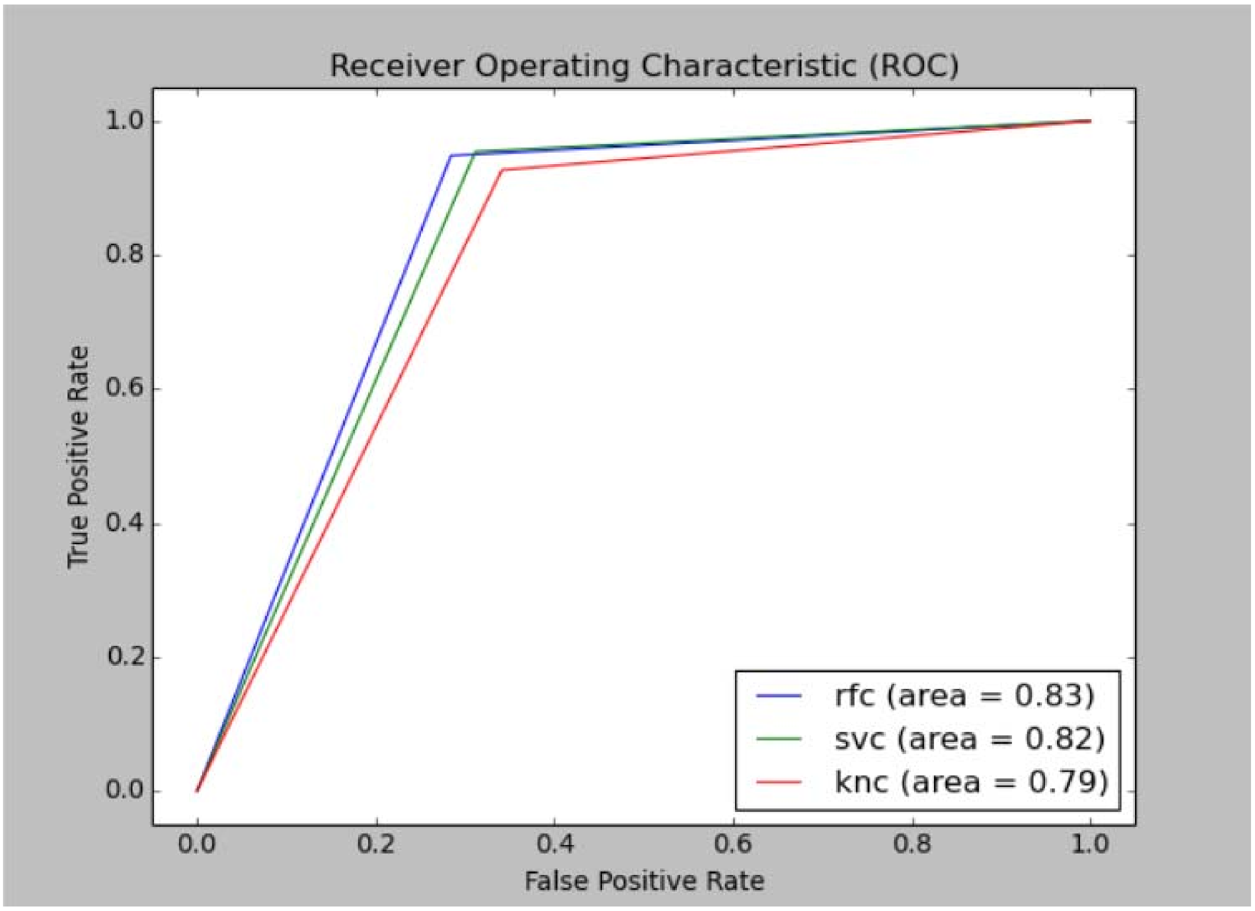

Figure 4 – ROC curve comparing the algorithms used on the given data set

Next, several classifiers were trained, using grid search and k-fold cross-validation (GridSearchCV in sklearn). Once several classifiers were trained, the results of the classifiers on the test data set were plotted and compared using an ROC curve. From this curve, it was determined that Random Forest Classifier was the best model to use for this problem, although the Support Vector Machine model also produced good results.

Finally, the Random Forest Classifier Model was refined by adding additional search parameters to the search space and re-running the grid search and k-fold cross-validation procedure.

The most difficult part of the project was determining which parameters to add to the search space for the grid search, as it is simply not feasible to add all of the available parameters. In addition to the selection of the parameters to search, it was necessary to come up with a set of possible values for the parameters. In some cases, this was easy – for example, in the RandomForestClassifier model's "criterion" parameter, there were only two possibilities ('gini' or 'entropy'). And, of course, binary parameters only had two possible choices as well. But for others, like 'n_estimators' for RFC which take on integer values, the range of values could be much larger. Since the grid search is exhaustive, adding even a few values increases the search time dramatically. So it was difficult to select a good range – a lot of trial and error was involved in the selection process.

The final model, while falling just shy of the benchmark value for ‘excellent’, did turn out to be pretty good. With additional work in parameter tuning and perhaps in pre-processing, it doesn’t seem unreasonable to believe that the classifier could be made ‘excellent’ by the defined benchmark standards.

Improvement

The best way to improve upon the results obtained in this project would be to implement the code in a much faster language, like C, so that the parameter space can be further mined for optimal values. Python is a great prototyping language, but the run times for GridSearchCV increase very quickly when adding even just a few new parameters to search. It would be possible to take the results obtained in this project as a guide to do a C implementation. This would, of course, be a time-consuming task, as it would involve implementing the algorithms directly from scratch in C (unless a library could be located that has already done the implementation).

It might make sense, and could possibly lead to an improved trainer, to attempt different strategies for outlier detection and removal. For example, the current project removes data points that have outlier values in 4 or more columns; perhaps it would help to remove data points that have outlier values in 3, 2, or even 1 column. In addition, an outlier value was considered to be one that fell outside of three standard deviations from the mean – removing outliers that fall outside, for example, 2.58 standard deviations (which corresponds to a 99% confidence interval), or even 1.96 standard deviations (a 95% CI) might help in the training and classification.

In order to improve the cross-validation strategy, which was k-fold, it might be helpful to run several iterations of the cross-validation using varying values of k for the k-fold in the grid search. It could be that more or less folds will result in better cross-validation of the grid search, leading to a better parameter selection. The problem with larger k is the amount of time the training takes, but with a C implementation (see above) this might not be as big an issue. In addition, it might be profitable to attempt some more model-specific cross-validation strategies in the grid search that allow for an entire range of values for certain parameters to be searched at once, rather than one value at a time.[21] This would allow a larger range of values to be searched, which could result in a better model. More research would need to be done to determine if this is appropriate for the models used in this project.

Finally, one algorithm that was not attempted, but that might very well perform better than the ones tried in this project, is the Neural Network. Neural Networks are very powerful algorithms, but also very complex algorithms. It was felt that they were probably more complex than was required for a good solution to this problem. However, since Neural Networks are more powerful, it is certainly feasible that the new benchmark achieved in this project using the Random Forest Classifier might be eclipsed by a Neural Network implementation.[22]

[21] http://scikit-learn.org/stable/modules/grid_search.html [section 3.2.4.1: Model-specific cross-validation]
[22] https://visualstudiomagazine.com/articles/2015/08/01/neural-network-binary-classification.aspx