Star-Galaxy Separation in DES with Machine Learning Algorithms

Avery Metzcar

May 24, 2023

Abstract

Machine learning algorithms are a prominent method of object classification in large astronomical surveys. With Python, there exists a wide variety of possible machine learning based classifiers to use. In this project, I experimented with nine different machine learning algorithms, seven from `scikit-learn` and two from `keras`. These were trained, in their default parameters settings or on 20 epochs, for the `keras` neural networks, on five different sets of input training data, with the classifier reset for each round of training to allow for comparison between the results. To measure the performance of these classifiers on the various sets of data, I created receiver operating characteristic (ROC) curves using `scikit-learn`, plotting the false positive rate of the classifier against the true positive rate. Additionally, I computed the areas under the curves (AUCs) for each round of training for each classifier, again with `scikit-learn`. From the results of this project, it is clear that using unique input features overall improves the performance of the classifiers and that using input data of low-dimensionality significantly inhibits the performance of the classifier and its potential optimization—although this was a recognized possible difficulty when beginning this project that I chose to maintain for the benefits of low computation time.

1 Introduction

Classification of observed objects is an important, yet tedious, fundamental step of astronomical research. An object cannot effectively be used to study galaxies if one is unsure if it is a galaxy. Visually classifying objects is time-consuming, which motivates pursuit of alternative methods of classifying large data sets. One promising method is the use of machine learning algorithms. Classifiers of this type can be trained on previously classified data and be used to predict the classification of newly observed objects, to varying degrees of accuracy. There are numerous machine learning methods available, with Python packages that provide them. Typical Python packages used in machine learning are `scikit-learn` and `keras`, which additionally may use other packages such as `numpy`. These allow for machine learning projects to be more efficient, as an individual does not need to write and design the function in order to implement it. Instead, one can use the pre-made algorithms available and attempt to optimize them, whether through changing the data or the algorithm's parameters.

2 Data Set

The data set used in this project was created for Sevilla-Noarbe et. al's 2018 paper on "Star-galaxy classification in the Dark Energy Survey Y1 dataset." It is composed of Dark Energy Survey (DES) observations taken with the DECam instrument mounted on the Victor M. Blanco 4-meter Telescope at the Cerro Tololo Inter-American Observatory in Chile. The fundamental purpose of DES was to study dark energy at various cosmological epochs; however, the quality and depth of the survey allows for it to have much broader astronomical applications. The imaging component of DECam is composed of 74 charged-coupling devices (CCDs), which allows high quality astronomical imaging and also reads faster than most other astronomical CCD cameras. With these components, DES covers 5000 square degrees of the sky in five photometric bands, *grizY*, with magnitude limits of (24.6,24.4,23.7,22.7,21.5), respectively, for 2 arcsecond apertures.

The data used for their project was a subset of the best quality data from DES SV and Y1, which formed the Gold catalog. Coadd images were used, so only objects for which more than one exposure in each band

was taken are included in the data set. Sevilla-Noarbe et. al, for their 2018 paper, obtained the catalogs by applying `SExtractor` (Bertin & Arnouts 1996) to each coadd image and calibrated the magnitudes using a global calibration module (Tucker et al. 2007). Afterward, they adjusted these by fitting them to the stellar locus in the i-band (High et al. 2009), which included correction of extinction.

The dataset used in this project, one of the three created for Sevilla-Noarbe et. al's 2018 paper, includes DES observation taken over the COSMOS field, which allowed for the true classification of the objects to be known. In this project, inspired by the previously mentioned paper, features used per each round of training were `MAG_AUTO_I` and `MAG_PSF_MOF_I`, `MAG_AUTO_I` and `SPREAD_MODEL_I`, and a two-component `PCA`-created data set from all feature columns of the data, meaning it excluded the image IDs, position measurements, and corresponding errors of the features. Descriptions of the three aforementioned features:

- `MAG_AUTO_I`: magnitude estimation in the i-band for an elliptical model based on the Kron radius.
- `MAG_PSF_MOF_I`: magnitude estimation in the i-band based on a PSF estimated model.
- `SPREAD_MODEL_I`: shape measurement in the i-band; morphological classification based on comparison between PSF models.

I used these features in my project because in the paper that served as inspiration for this project, Sevilla-Noarbe et. al 2018, they used `MAG_AUTO_I`, `FLUX_RADIUS_I`, and `SPREAD_MODEL_I` to train their best performing machine learning algorithm, a support vector machine. From my understanding, the choice of i-band based features is due to the low dependence of luminosity on stellar age and metallicity in that band. In my initial round of training, I used `MAG_AUTO_I` and `MAG_PSF_MOF_I` as I could not find `FLUX_RADIUS_I` within the data set and was unsure how Sevilla-Noarbe et. al obtained it. Additionally, my choice of `MAG_PSF_MOF_I` was rooted in a misunderstanding of what `SPREAD_MODEL` was and its applicability to classification with machine learning as I had previously understood it as a classification and not a shape measurement. After this misunderstanding was corrected, I retrained my algorithms using `MAG_AUTO_I` and `SPREAD_MODEL_I` to obtain results that were more relevant to the original paper.

3 Machine Learning Algorithms

Nine supervised machine learning algorithms were used in this paper: three tree-based, three neural networks, and three other classifiers. The purpose of this was to explore the advantages and disadvantages of a wide-range of classifiers on this data set and possible methods to improve the performance. These algorithms were initially used with their default scikit-learn input values, excluding two of the neural network-based classifiers, which I built using `keras` and trained for 20 epochs each.

3.1 Tree-based Classifiers

Tree-based classifiers used in this project were scikit-learn's `DecisionTreeClassifier`, `ExtraTreesClassifier`, and `RandomForestClassifier`. Decision trees like that of `DecisionTreeClassifier` are the fundamental components of ensemble algorithms like `ExtraTrees` and `RandomForest`, and they categorize the data according to its features until the max depth of the tree is reached. In other words, it begins with one node at the input, which divides into two, which divide into two, and continues this process until the pre-specified depth is reached. In its default setting, `scikit-learn`'s `DecisionTreeClassifier` has no specified depth, causing it to divide until all of the "leaves" are pure. This often leads to overfitting for the classifier and creates very complex trees, as seen in the Appendix. To avoid the disadvantages of decision trees, ensemble methods like `ExtraTrees` and `RandomForest` exist. The difference between these two methods is that `ExtraTrees` divides the nodes of its various trees based on randomly selected features, while `RandomForest` divides based on learned features of the data. For both of these classifiers, the outputted classification of an object with a given set of features is the one with the most "votes" from the decision trees that compose the classifier. Typically, `RandomForest` is the preferred algorithm of these three.

3.2 Three Neural Network-based Classifiers

The neural network based classifiers used in this project were two neural networks built using `keras`, one categorical and one binary, and `scikit-learn`'s multi-layer perceptron classifier, `MLPClassifier`. For the `keras` neural networks, the model summaries are displayed below. These two neural networks are identical except for the activation function in the final layer and the overall loss function. The binary neural network has a sigmoid final activation function, meaning that it is logistic and outputs values between 0 and 1. Its binary cross-entropy loss function restricts the network to only being able to be applied to two-class data. The truth labels inputted into the `keras` neural networks was one hot encoded, which may have affected the binary neural network's performance; however, since there were only two possible classifications, it was able to classify the data, although potentially not as accurately as it would if the truth labels were not categorical. The final layer of the categorical neural network's activation function was softmax, which is probabilistic and outputs values between 0 and 1 that together add up to 1. Correspondingly, the loss function of this neural network was categorical cross-entropy. For both of these neural networks, the loss function, whether it was binary cross-entropy or categorical cross-entropy, is how the neural network learns over multiple epochs.

The final neural network used was `scikit-learn`'s `MLPClassifier`, which is a basic neural network with a hidden layer of a given number of neurons. In its default state, this hidden layer is composed of 100 neurons. `MLPClassifier` iterates over the data until convergence is reached within a given tolerance or until the given maximum number of iterations is reached, for which the default is 200. This also means that overfitting is likely when using `MLPClassifier` in its default state, which will affect its performance on test data. Alternatively, I trained the two keras neural networks for 20 epochs. Overfitting may have also occurred here, most notably for the binary neural network, as over the course of the project it became clear that it became overfitted to the data more quickly than the categorical neural network. Although the possibility of overfitting must be accounted for in judging the performance of any machine learning classifier, it is especially notable with neural networks.

3.3 Three Additional `scikit-learn` Classifiers

The other three classifiers used do not have similar fundamental components, but they are all classifiers provided by `scikit-learn`. These three are a categorical Naive Bayes classifier, `CategorialNB`, a k-nearest neighbors classifier, `KNeighborsClassifier`, and a stochastic gradient descent classifier, `SGDClassifier`. `CategorialNB` is a simple probabilistic classifier based on Bayes' theorem. It was only used in the unprocessed data rounds as I discovered that it only took positive x input data, for which the standardized data did not comply. The other two classifiers in this set of three were both able to be trained on unstandardized and standardized data. `KNeighborsClassifier` computes the distance between the test data and the training points and restricts to the "k" number of points closest to the test data whose classes serve as "votes" of which class the test data point belongs to. `scikit-learn`'s default k value is five, so five close training data points are used to classify a test data point. Although in its default setting all of the points are unweighted, adding weight to some of the data points, along with changing the k-value, can help to optimize `KNeighborsClassifier`'s performance. Lastly, `SGDClassifier` applies a basic stochastic gradient descent gradient learning routine to the data and minimizes a given cost function; in its default `scikit-learn` setting, `SGDClassifier` functions like a linear support vector machine (SVM).

4 Results

The classifiers were trained on five types of data, with the kernel restarted each round to allow for proper comparison. This was to study the impact of the input features on the performance of the classifier.

4.1 Unstandardized `MAG_AUTO_I` and `MAG_PSF_MOF_I`

In the first round, the input features were the `MAG_AUTO_I` and `MAG_PSF_MOF_I` columns of the DES COSMOS dataset. No preprocessing occurred prior to training. The resulting ROC plots are below in Figure 1. Notable features of these graphs are that on the default parameter settings with this input data, `scikit-learn`'s

tree-based classifiers appear to have the best ROC curves. Additionally, the binary neural network performed better than the multi-layer perceptron classifier and the categorical neural network.

4.2 Standardized MAG_AUTO_I and MAG_PSF_MOF_I

Then, after restarting the kernel, the input features were standardized using `scikit-learn`'s `StandardScaler`, which removes the mean of the data and scales each element to one unit variance. Standardization is a typical preprocessing step of machine learning training and is generally expected to improve the performance of the algorithms. As one can see from Figure 2, the ROC curves are overall much closer together with standardized data in comparison to the unstandardized data. In these plots, the best performing classifiers appear to be the multi-layer perceptron, extra trees, random forest, and k-nearest neighbors. Also, all of the neural network-based algorithms improved notably, although, with the standardized data, the binary neural network performs worse than the other two. As described previously, this is likely due to overfitting, as I used the same number of epochs (20) for both the categorical and binary neural networks, and the inputted classifications (the y-inputs) being one hot encoded. Nonetheless, it is interesting to see how the two neural networks perform in comparison to each other.

4.3 Unstandardized MAG_AUTO_I and SPREAD_MODEL_I

After resolving the misunderstanding that I had had previously (as addressed in section 2 of this paper), I repeated the process of the last round—this time, inputting `MAG_AUTO_I` and `SPREAD_MODEL _I` as the features. This allows for the training to be more similar to the inspiring paper. With these features, unstandardized, the binary neural network clearly excels in comparison to the others. Its ROC curve in Figure 3 is better than its curve with the standardized features in the previous round in Figure 2. In contrast, the stochastic gradient descent classifier and the categorical Naive Bayes one perform notably worse with this data. This is likely due to the simplicity of these classifiers and the stark differences between the two input features. Although many of the other classifiers perform better with two very different features, one magnitude and one shape measurement, `scikit-learn`'s `SGDClassifier` and `CategorialNB` performed better when the input features were both magnitudes and therefore similar in scale.

4.4 Standardized MAG_AUTO_I and SPREAD_MODEL_I

Then, as in the previous round, the data was standardized with `StandardScaler` and the classifiers retrained. As displayed in Figure 4 below, it is visibly difficult to discern the best performing classifier. All of the classifiers, excluding `CategorialNB`, which could not be used on the standardized data due to negative values, and `SGDClassifier`, performed better with the standardized data of Figure 4 than the unstandardized data of Figure 3. Additionally, the ROC curves of this round are an improvement of those of the standardized magnitude-feature round. From the comparison of the results of the magnitude-feature round and the magnitude-and-shape-feature round, it is evident that it is overall better for machine learning classification to utilize two unique, contrasting features rather than two similar ones of the same type.

4.5 2-Component PCA Input Data

The last set of data used was produced using `scikit-learn`'s principal component analysis function (`PCA`), which linearly reduces the dimensionality of the input data. To use this, I created an array with all the feature columns of the data—therefore ignoring irrelevant ones for this purpose such as RA and DEC—standardized this array using `StandardScaler`, which is necessary for `PCA` use, and inputted this array into the function with a set number of components of 2. This was to maintain the number of input dimensions of previous rounds, allowing for comparison to be more appropriate. As seen in Figure 5, trained on this data, the categorical neural network clearly performed the best, with many of the other classifiers close behind. Another notable feature of these graphs is that the binary neural network has fallen in comparison to the other algorithms, as its ROC curve, which was relatively good in both sets of unstandardized data, is now the second worst, with only the stochastic gradient descent classifier performing worse than it. Nonetheless, all of the algorithms have visibly improved between the unprocessed, magnitude-feature data and the `PCA` data. It is difficult to visually discern differences between the unique-feature data and the `PCA` data.

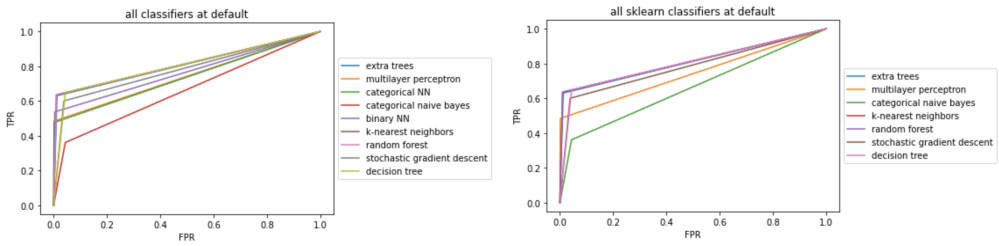

Figure 1: ROC curves of classifiers with unstandardized `MAG_AUTO_I` and `MAG_PSF_MOF_I` input data.

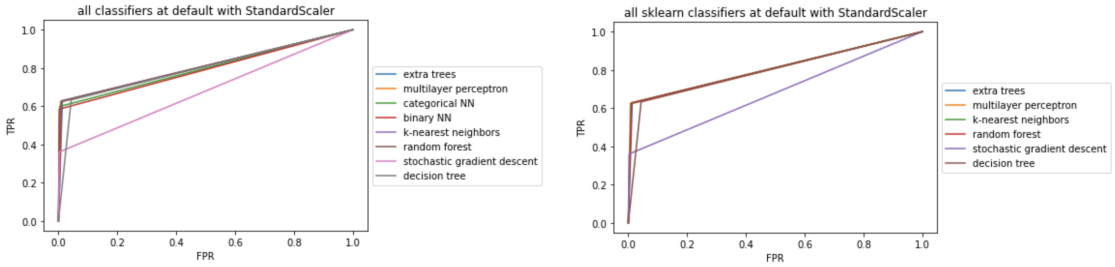

Figure 2: ROC curves of classifiers with standardized `MAG_AUTO_I` and `MAG_PSF_MOF_I` input data.

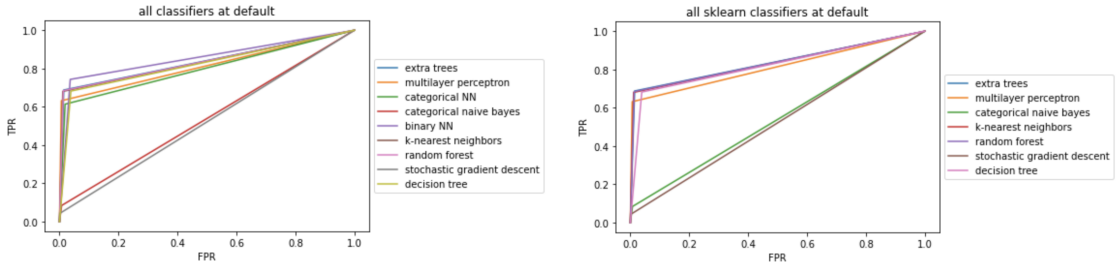

Figure 3: ROC curves of classifiers with unstandardized `MAG_AUTO_I` and `SPREAD_MODEL_I` input data.

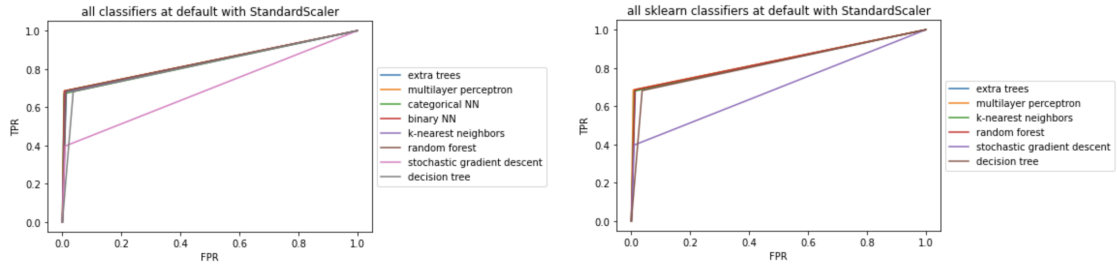

Figure 4: ROC curves of classifiers with standardized `MAG_AUTO_I` and `SPREAD_MODEL_I` input data.

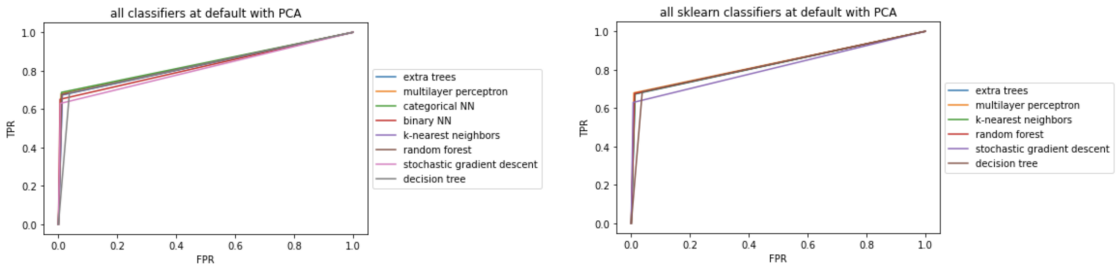

Figure 5: ROC curves of classifiers with 2-component `PCA`-created input data.

4.6 Areas Under the Curves

With the actual areas under the curves (AUCs) displayed in Table 1, it is more clear how each classifier performed over the course of these trials. "Mag-Feature Ust" corresponds to the unstandardized magnitude-feature data, while "Mag-Shape-Feature Ust" corresponds to the unstandardized unique-feature data. In all rounds other than that of the unstandardized magnitude-feature data, the neural network-based classifiers performed the best. Interestingly, the highest AUC of all the rounds is from the binary neural network with the unstandardized unique-feature data, and this neural network remained one of the highest performers when this data was standardized as well. Another notable aspect of this table is that there is a larger progression over the rounds of magnitude-feature data to the `PCA` round than there is over the unique-feature rounds to the `PCA` round. This indicates that for the purposes of this classification, the use of two unique features, one magnitude and one shape, causes overall better classifier performance than the use of two magnitude features.

Table 1: AUC values for each classifer in each round, where USt=unstandardized and St=standardized.

Classifier	Mag-Feature USt	Mag-Shape-Feature USt	Mag-Feature St	Mag-Shape-Feature St	2-Component `PCA`
ExtraTreesClassifier	0.809294	0.836569	0.805831	0.836471	0.830222
MLPClassifier	0.741635	0.812591	0.808809	0.839614	0.835192
Categorical NN	0.737523	0.797248	0.797408	0.832922	0.838384
CategoricalNB	0.658990	0.538415	N/A	N/A	N/A
Binary NN	0.766728	0.853289	0.791101	0.838949	0.822815
KNeighborsClassifier	0.813053	0.834620	0.809043	0.834498	0.831204
RandomForestClassifier	0.812642	0.835655	0.808324	0.836879	0.832490
SGDClassifier	0.781227	0.520889	0.679033	0.693801	0.811558
DecisionTreeClassifier	0.802445	0.821635	0.794808	0.822615	0.823809

5 Optimization

After these various data trials, I attempted to optimize the best performing classifiers, such as `MLPClassifier`, `RandomForestClassifier`, and the `keras` categorical neural network. For the `scikit-learn` classifiers (`MLPClassifier` and `RandomForestClassifier`), I used `scikit-learn`'s `GridSearchCV` and `HalvingGridSearchCV` functions. `GridSearchCV` automates searching for the best performing parameters from a given parameter grid. To use, one inputs the classifier of choice and a dictionary of parameter values to test over. For example, for `MLPClassifier`, I inputted a parameter grid of 'hidden_layer_sizes': [100,150,200], 'activation': ['identity', 'logistic', 'tanh','relu']. Then, I fitted the resulting `GridSearchCV` object to the training data.

However, GridSearchCV is exhaustive, fitting all of the data for each combination of settings of the classifier. If a classifier takes at around 2 minutes fitting to the training data in its default parameter settings, like MLPClassifier does, and one inputs the parameter grid that I did, that means that GridSearchCV is performing 60 fits that may take around 2 minutes each—although the amount of time a fit takes is primarily dependent on the parameters. With this parameter grid for MLPClassifier, GridSearchCV took 25 minutes to run and determine the best performing parameters. As a result of the exhaustive quality of GridSearchCV, scikit-learn also provides an experimental tool, HalvingGridSearchCV, which functions similarly to GridSearchCV but uses part of the data rather than the entirety of the inputted data set for each of the, in this case, 60, parameter fits. Generally speaking, GridSearchCV and HalvingGridSearchCV output the same results, making HalvingGridSearchCv a preferable mode of optimization of scikit-learn machine learning algorithms.

Additionally, I used 3-component PCA-created data in my optimization for two reasons: 1. The use of the PCA improved overall performance of the classifiers, although some may have performed slightly better (within 0.03 of the areas under the curves scores) and 2. In Sevilla-Noarbe et. al's 2018 paper that this project is based off of, they used three input features in their best performing machine learning classifier. Previously, I used a 2-component PCA to maintain consistency with the other rounds of data training; however, it was clear that increasing the dimensionality of the data would be feasible in terms of computation time and likely improve classifier performance.

For the MLPClassifier, HalvingGridSearchCV did output a different set of best parameters than GridSearchCV, which was unusual and did not occur in my other uses of these tools. HalvingGridSearchCV outputted best parameters of a hidden layer of 100 neurons with the tanh activation function, while GridSearchCV outputted best parameters of 150 neurons with the tanh activation function. These had corresponding AUCs of 0.827251 and 0.836907, in contrast to the 0.835192 achieved with the 2-component PCA data with the classifier's default parameters of a 100 neuron hidden layer with the relu activation function. In its default parameter settings of 100 neurons with the relu activation function and the 3-component PCA data, the area under the curve is 0.832035. The AUC of the 3-component PCA data is improved with the GridSearchCV best parameters; however, it is notable that the 3-component PCA data AUC with default parameters is worse than that of the default parameters and 2-component PCA data. However, since the two values are within 0.005 of each other, this discrepancy alone is not sufficient to make claims about the "better" dimensionality or parameter set.

Another classifier that I tried to optimize is RandomForestClassifier. For this classifier, HalvingGrid SearchCV outputted best parameters of number of decision tree estimators of 250, a maximum depth of 10 for these trees, and the entropy criterion function to calculate the quality of each split. Since the two optimization search methods outputted different best parameters for MLPClassifier, I used GridSearchCV as well; however, with RandomForestClassifier and a parameter grid resulting in 90 possible fits, GridSearchCV required nearly twenty times as much time as HalvingGridSearchCV to output the same best parameters. With these parameters and 3-component PCA data, the AUC score of RandomForestClassifier is 0.825228 in comparison to its score of 0.832490 with the default parameters and 2-component PCA data. In its default parameter settings with the 3-component PCA data, the area under the curve is 0.833268. In contrast to the MLPClassifier results, the increased dimensionality of the PCA data did improve the AUC with the classifier's default parameter settings. However, there is not notable improvement between the default parameters and the GridSearchCV outputted parameters. I think that the minimal optimization occurring is due, at least in part, to the relatively low-dimensionality of the input data. A higher number of PCA components may improve the results.

Lastly, I varied the epoch values of the keras categorical and binary neural networks that I built, training on the 3-component PCA data. With the original number of 20 epochs, the AUC scores of the categorical and the binary neural networks are 0.830879 and 0.841509, respectively. I tested numerous epoch numbers and paid attention to the validation accuracy that it outputs for each epoch to gauge an ideal number of epochs. The most efficient approach to optimizing the number of epochs is to start with a low value, then increase the number of epochs by retraining the same model for some set time. The best AUC score I found for the categorical neural network with 3-component PCA data was 0.849419, with 17 epochs of training. One difficulty in attempting to optimize the number of training epochs is that neural networks of the same architecture never perform the same each time. In one attempt, the neural network may output an AUC of 0.84, but in the next, reset, with the same number of epochs, it may be 0.81. Of course, with all machine

learning algorithms, the performance varies, but I have found it to be especially notable with neural networks. For the binary neural network, the best AUC score with the 3-component PCA data was 0.843382, trained on 17 epochs like the categorical one. This is likely very coincidental. As previously noted in this paper, the binary neural network tends overall to get overfitted faster than the categorical one, so although the best performance happened to occur at the same number of epochs, generally speaking, one would reasonably expect the binary neural network to perform better with less epochs than the categorical neural network.

6 Discussion

From this project, it is clear that scikit-learn provides many promising machine learning classifiers applicable to classifying stars and galaxies based on inputted features. An ideal next step for this project would be to improve the performance of these classifiers through optimization of their parameters and determining the most beneficial input data. As seen in Figure 7 in the Appendix—which is Figure 1 of the previously mentioned 2018 paper by Sevilla-Noarbe et. al—in reference to those displayed in the Results section, there is notable room for improvement in the ROC curves of the classifiers used within this project.

To focus on potential improvements of the inputted data, the most promising classifiers improved from choosing the two features to be unique in comparison to each other in contrast to their performance when the two input features were magnitudes. This makes logical sense as it allows for more definable characteristics for a classifier to learn to distinguish between. Additionally, the shape measurement inputted (SPREAD_MODEL_I) provides a more clear foundation for classification than magnitudes as the DES information for this data set notes that it is a morphological classification with certain values being associated with galaxies and others with stars. Most of the classifiers performed comparatively well with the input features of MAG_AUTO_I and SPREAD_MODEL_I as it did with the 2-component data created using PCA. Due to my previously mentioned misunderstanding of the applicability of SPREAD_MODEL_I to machine learning classification, certain parts of this project occurred in a less than ideal order. Now knowing the performance of the classifiers with the MAG_AUTO_I and SPREAD_MODEL_I input features, I cannot say that PCA-use is a necessary component for this type of classification, although it did improve the performance to a comparable level of the unique input features in comparison to the magnitude features.

Another limitation of the data used in this project is the low-dimensionality of the input data. This was known prior to any training and was a conscious choice due to computation time. Nonetheless, it is clear that future trials would benefit by increasing the number of features inputted to the classifiers. As seen by the similar performance between MAG_AUTO_I and SPREAD_MODEL_I and the 2-component PCA data, it would be interesting to discover if there exists a point at which a number of unique features performs notably better or worse than PCA-created data of the same number of components.

This low-dimensionality of the inputted data significantly inhibited the potential optimization of the classifiers as well. As noted in the Optimization section of this paper, no notable improvement occurred in the performance of the classifiers utilizing "best" parameters found by scikit-learn's GridSearchCV or HalvingGridSearchCV. In other projects, with higher-dimension input data sets, I have seen significant improved by using the best parameters that these tools find. As a result, the issue is very clearly with the low-dimensionality of my input data. Future attempts to optimize these classifiers (or the best performing of them) for this type of classification should seek to find a ideal balance between input dimensions, classifier parameters, and computation time. This only applies to the scikit-learn classifiers, although for the neural networks built using keras, similar logic applies. For these, rather than searching for the best performing set of parameters, the optimization could occur in experimenting with other network architectures and epoch numbers. As seen from the model summary in the Appendix, the architecture of these two neural networks was quite simple, and there are many possible changes that could be made, such as changing the number or type of layers, changing the number of neurons within the layers, or experimenting with different activation functions for these. Additionally, it would be interesting to experiment with more types of classifiers to see if any perform better than what has currently been found. After finding the best performing set of algorithms and input features, the next step would be to apply these to unclassified DES objects, as done in Section 5 of Sevilla-Noarbe et. al's 2018 paper.

7 Appendix

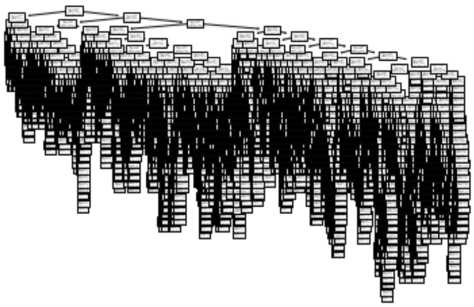

Figure 6: Visualization of the decision tree formed by `scikit-learn`'s `DecisionTreeClassifier` trained on this data set on its default parameter settings.

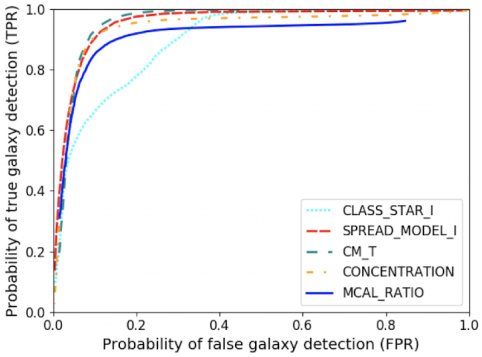

Figure 7: ROC curves of non-ML classifiers from Sevilla-Noarbe et. al's 2018 paper.

Layer (type)	Output Shape	Param #
dense_3 (Dense)	(None, 30)	90
flatten_1 (Flatten)	(None, 30)	0
dense_4 (Dense)	(None, 30)	930
dense_5 (Dense)	(None, 2)	62

Total params: 1,082
Trainable params: 1,082
Non-trainable params: 0

Figure 8: Model summary of the neural networks to demonstrate the general layer types that compose them, although details such as activation functions and loss functions are missing. These were describe in the Machine Learning section of the paper.

8 References

Bento, C. (2021, Sept. 21). Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis. https://towardsdatascience.com/multilayer-perceptron-explained-with-a-

real-life-example-and-python-code-sentiment-analysis-cb408ee93141.
Christopher, A. (2021, Feb. 2). K-Nearest Neighbor. https://medium.com/swlh/k-nearest-neighbor-ca259 3d7a3c4#:~:text=K%2Dnearest%20neighbors%20(KNN),closet%20to%20the%20test%20data.
Koech, K. (2020, Oct. 2). Cross-Entropy Loss Function. https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e.
McQuinn, K., et. al. (2019). Using the Tip of the Red Giant Branch As a Distance Indicator in the Near Infrared. https://iopscience.iop.org/article/10.3847/1538-4357/ab2627
Naveen. (2022, April 20). Difference between Sigmoid and Softmax activation function?. https://www.nomi dl.com/deep-learning/what-is-the-difference-between-sigmoid-and-softmax-activation-function/#:~:text =Sigmoid%20activation%20function%20is%20a,output%20into%20a%20probabilistic%20one.
neuralthreads. (2021, Dec. 1). Categorical cross-entropy loss — The most important loss function. https://neuralthreads.medium.com/categorical-cross-entropy-loss-the-most-important-loss-function-d37 92151d05b.
Sameer. (2021, May 9). Naive Bayes Classification Algorithm in Practice. https://python.plainenglish.io/ naive-bayes-classification-algorithm-in-practice-40dd58d18df4.
Sevilla-Noarbe, I., et. al. (2018, May 7). Star-galaxy classification in Dark Energy Survey Y1 dataset. https://arxiv.org/abs/1805.02427.
Srinivasan, A. (2019, Sept. 6). Stochastic Gradient Descent — Clearly Explained!!. https://towardsdatasc ience.com/stochastic-gradient-descent-clearly-explained-53d239905d31.
Thankachan, K. (2022, Aug. 6). What? When? How? ExtraTrees Classifier. https://towardsdatascience.com/what-when-how-extratrees-classifier-c939f905851c.
Yiu, T. (2019, Jun. 12). Understanding Random Forest. https://towardsdatascience.com/understanding -random-forest-58381e0602d2.
(n.d.). https://www.darkenergysurvey.org/the-des-project/instrument/.
(n.d.). https://des.ncsa.illinois.edu/releases/y1a1/gold/classification.
*Additionally, information available for these classifiers and tools on `scikit-learn`'s website was referenced for this project.