Draft version June 13, 2019
Typeset using LaTeX **modern** style in AASTeX62

Comparing Source Classifiers on SDSS Color Indices

Preston Went[1]

[1] *University of Washington, Seattle*
Department of Astronomy

ABSTRACT

With the advent of automated telescope surveys, like the Sloan Digital Sky Survey (SDSS) and upcoming Large Synaptic Survey Telescope survey (LSST), it is no longer feasible for human astronomers to look at every photo and spectrum taken. These unseen data hold new insights. To find them, it is necessary to understand the theory and application of machine learning techniques in an astronomical setting. Here, I explore different methods for supervised source object classification on color index data from SDSS with the star, galaxy, and quasar (QSO) labels. For this dataset and task, I argue a k-Nearest Neighbors classifier offers the best usability, while a Kernel Support Vector Machine offers the best performance and features.

Keywords: methods: data analysis — methods: statistical

1. INTRODUCTION

The 15th data release (DR15) of SDSS contains images of nearly 500 million objects and spectra of nearly 5 million [Aguado et al. (2018)]. While massive data releases like this open new avenues of research – by capturing a significant number of rare objects, for instance – they also introduce a set of problems. One of the largest such problems is the inability of any human or reasonably sized group of humans to see the whole of the data. This complicates otherwise relatively simple tasks like classifying objects as stars, galaxies, or QSOs as they are observed.

Thankfully this problem, created by automation, can be solved by automation. The size of modern astronomical datasets allows for the proper application of modern statistical techniques – including machine learning, which I define here as the study of probabilistic algorithms for producing functions that approximate given data. Machine learning techniques are typically divided into two classes:

- Supervised, in which the data come with a label, either discrete or continuous, that one wishes to apply to yet unlabeled data.

Corresponding author: Preston Went
preston.went@outlook.com

- Unsupervised, in which the data are unlabeled, and the goal is to extract features from the data.

In the task of classifying sources into stars, galaxies, and QSOs – upon which the rest of the paper is focused – the application of machine learning involves a few basic steps:

1. Defining a feature matrix, X, and a label vector, y. This is often called "feature engineering".
2. Defining a model function, f, to make approximate to the data.
3. Fitting the model to the data, often called "training" the model. The model here is said to "learn" the data. This essentially involves using an algorithm, A, to produce a function $g = A(f, X, y)$ such that $g(X) \approx y$ but g has a domain larger than that defined by X.
4. Using the model to predict the labels for new data. This is termed "prediction".

In particular, the remainder of the paper is organized as follows:

- Section 2 introduces the data used for the remainder of the paper, explores some of its features, and describes how it was procured.
- Section 3 briefly introduces the various machine learning models and algorithms employed for the task of star, galaxy, QSO classification, and offers a hypothesis for each as to whether or not it produces good results on the data, and describes how those results are tested.
- Section 4 shows the results of training and testing the algorithms on the data, then compares and contrasts across models.
- Section 5 provides a concise synopsis of the all of the above.

2. DATA

Training and testing a classifier requires a set of pre-labeled data. In the case of this exploration, I use data drawn from SDSS DR15. I queried in the CasJobs system using the following commands:

```
SELECT TOP 300000
    p.objid, p.ra, p.dec, p.u,
    p.g, p.r, p.i, p.z,
    s.class into mydb.MyTable from PhotoObj AS p
```

```
    JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE
    p.u BETWEEN 0 AND 19.6
    AND s.class = 'star'
    AND s.zWarning = 0
    AND s.z between 0 AND 0.4
    AND p.clean = 1

SELECT TOP 50000
    p.objid, p.ra, p.dec, p.u,
    p.g, p.r, p.i, p.z,
    s.class into mydb.MyTable_0 from PhotoObj AS p
    JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE
    p.u BETWEEN 0 AND 19.6
    AND s.class = 'galaxy'
    AND s.zWarning = 0
    AND s.z between 0 AND 0.4
    AND p.clean = 1

SELECT TOP 50000
    p.objid, p.ra, p.dec, p.u,
    p.g, p.r, p.i, p.z,
    s.class into mydb.MyTable_1 from PhotoObj AS p
    JOIN SpecObj AS s ON s.bestobjid = p.objid
WHERE
    p.u BETWEEN 0 AND 19.6
    AND s.class = 'qso'
    AND s.zWarning = 0
    AND s.z between 0 AND 0.4
    AND p.clean = 1
```

These provide SDSS object IDs, right ascensions, declinations, and ugriz filter brightnesses for stars, galaxies, and quasars, respectively. They also filter out any object which suffered from observational difficulties by selecting only those without zWarnings and that have been declared clean, and remove objects with large redshifts ($z \geq 0.4$). They yielded 92 825 stars, 50 000 galaxies, and 10 130 QSOs.

Once procured, I resolved the issue of unknown luminosities for these objects by reducing the ugriz magnitude data to the u − g, g − r, r − i, and i − z color indices. The color space and position distributions of the data are given in Figure 1 and Figure 2, respectively, while the pairwise Pearson correlation coefficients of the color indices are given in Table 1.

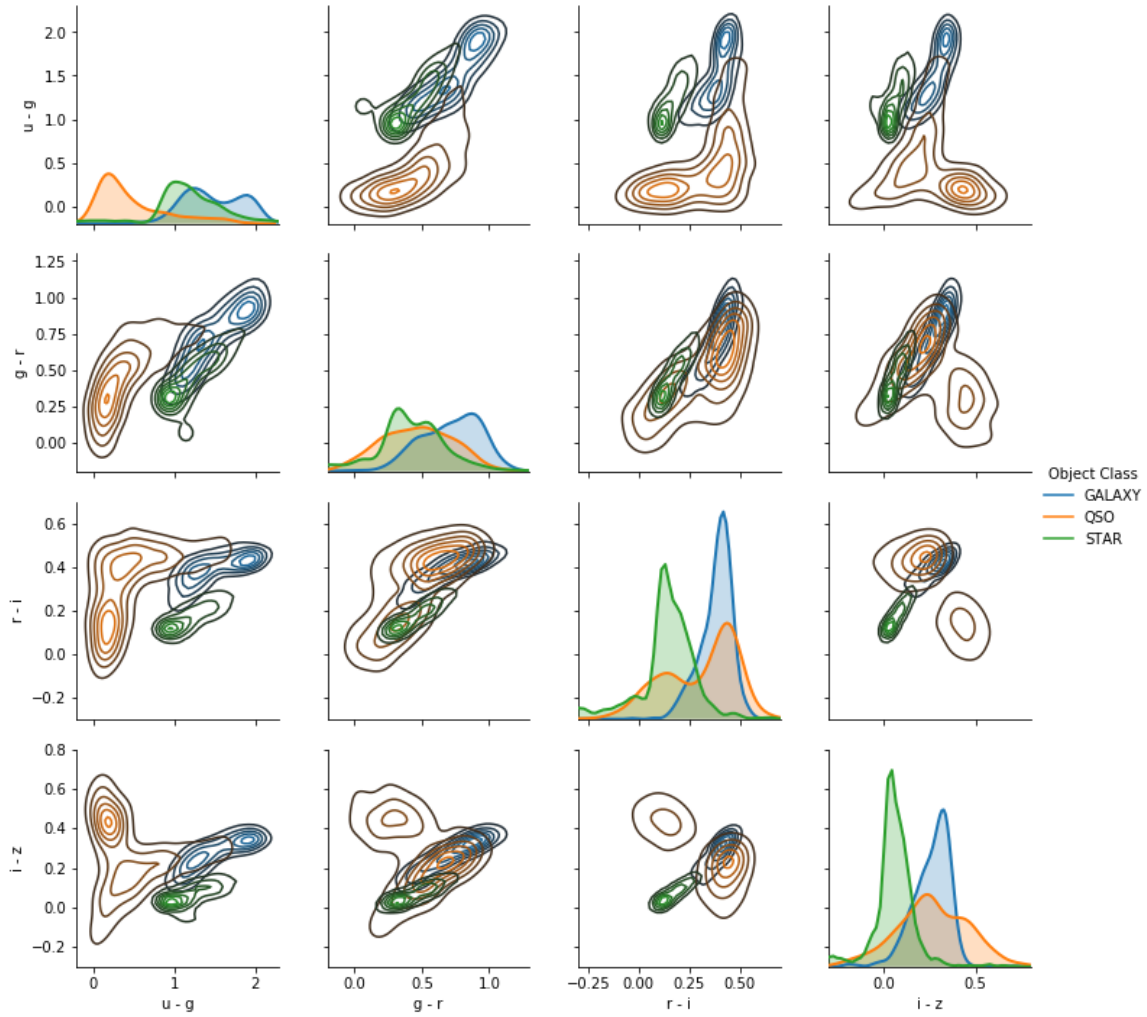


Figure 1. The color space distribution of the data in magnitudes. Distributions produced using kernel density estimation with Gaussian or bivariate Gaussian kernels.

	u − g	g − r	r − i	i − z
u − g	1.00	0.75	0.45	0.36
g − r	0.75	1.00	0.70	0.55
r − i	0.45	0.70	1.00	0.14
i − z	0.36	0.55	0.14	1.00

Table 1. The pairwise Pearson correlation coefficients of the color indices.

It is worth noting that QSOs seem to be separable from stars and galaxies in the u − g index, and stars and galaxies appear to be partially separable from each other in the r − i and i − z indices. Another interesting feature is that the color indices tend to be correlated with those near them.

Some machine learning methods are sensitive to the mean and variance of the data in each dimension, so the data here were scaled to unit variance and shifted to a null mean.

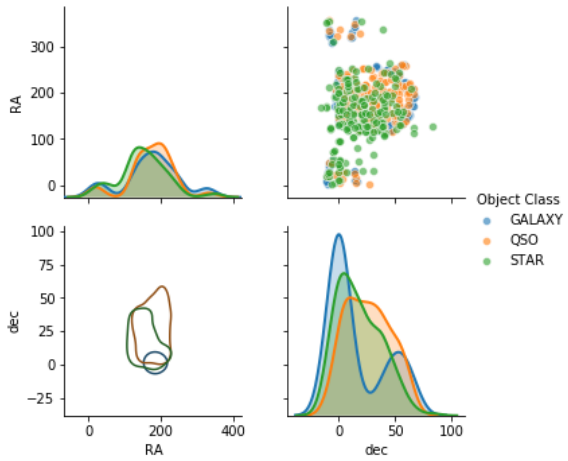


Figure 2. The right ascension (RA) and declination (dec) of the data in degrees. Distributions produced using kernel density estimation with Gaussian or bivariate Gaussian kernels.

3. DATA PROCEDURES

Having found a labeled dataset, removed from it the points with the greatest errors, and handled the major systematic errors associated with it, we may now discuss the classifiers I tested and how I tested them. I used classifiers and test methods as implemented in scikit-learn [Pedregosa et al. (2011)] in all cases, and much of the discussion here is paraphrased from the scikit-learn project website.

I will go over the classifiers first, then discuss the metrics I used to judge their performance.

3.1. *Classifiers*

3.1.1. *Gaussian Naive Bayes*

The Gaussian Naive Bayes (GNB) classifier is particularly simple. It makes the assumptions that the features of the dataset are uncorrelated, and that each class has a Gaussian distribution in each feature. Applying these assumptions to Bayes' rule, it then finds the parameters of the Gaussians for which the likelihood of the data is maximized. Prediction can then be performed by finding the label which best explains the position of the an unlabeled point in the feature space relative to the learned Gaussians.

GNB tends to be computationally inexpensive, and has very few hyperparameters to adjust. However, the assumptions made by GNB are rather strong. In this case, Figure 1 and Table 1 suggest that both of the major assumptions mentioned above are incorrect for this dataset. We should thus expect relatively poor predictive performance from this classifier.

3.1.2. *Kernel Support Vector Machine*

In some cases, classes may be separable in the feature space by a set of lower-dimensional hyperplanes. The support vector machine (SVM) selects a few points

near the edges of each class – the support vectors from which the method draws its name – and creates the hyperplanes that separate them with the largest possible margins. A kernel SVM casts the data into a higher dimensional space first using a kernel function, then creates the hyperplanes in this new space. This allows the SVM to model classes that are not linearly separable.

SVMs are memory efficient – needing only to store the support vectors that define the hyperplanes that separate the classes. They can also be modified to provide estimates during prediction for the probability that a point is in each class. However, SVMs are generally slower than the other methods discussed here, and the parameters of the resultant model are generally not interpretable in a physical sense.

A kernel SVM should generally perform quite well on this dataset, given that it is almost linearly separable anyways, but we should expect it to train a bit more slowly.

3.1.3. *Linear Discriminant Analysis*

Linear discriminant analysis (LDA) is another application of Bayes' rule, like GNB, this time assuming that the probability of belonging to each class follows a multivariate Gaussian distribution and that the covariance matrix of each class is the same. This creates linear decision surfaces between the classes. Here we have dropped the assumption that the features are uncorrelated, and we no longer require that the classes are precisely Gaussian in each feature.

Like GNB, LDA tends to be comparatively inexpensive and quick. There are also few to no hyperparameters to adjust.

Having said this, we expect LDA to train quickly and produce better results than GNB. The data appear almost linearly separable, so we expect LDA's performance to be not substantially less than the kernel SVM's.

3.1.4. *Quadratic Discriminant Analysis*

Quadratic discriminant analysis (QDA) is nearly the same as LDA, with the exception that the multivariate Gaussians that are fit to the classes can have different covariance matrices. Relaxing in this way produces quadratic decision surfaces between the classes.

We thus expect QDA to train at a rate similar to LDA, and perform similarly. The difference in performance between LDA and QDA can be used to measure the degree to which the data are linearly separable.

3.1.5. *k-Nearest Neighbors*

k-nearest neighbors (KNN) operates simply. To predict the label for a new sample with this method, one simply finds the k training samples that are nearest it using some metric – typically the Euclidean distance – and adopts the most common label from that subset. The choice of k is often optimized using a guess-and-check option. Here, I chose $k = 5$.

Calculating distance is a simple operation, so we expect KNN to be fairly quick. It is, however, dependent upon the distance between points, so we must have a sufficient density of points throughout our feature space for it to work. What is "sufficient" is not well-defined, though a good general rule is to have substantially more samples in the training set than there are features. Given that we have around 150 000 samples and only four features, this should not be an issue.

3.1.6. *Decision Tree*

A decision tree (DT) is similar to KNN in that it has no parameters and is quite quick. That is where the similarity ends. DTs model the data using a set of logical decision rules, constructing the rules to perfectly fit the training data.

As the rules are constructed to perfectly fit the training data, DTs can be highly sensitive to the choice of training data. DTs also tend to overfit, producing high variance models that may not accurately reflect the truth. So long as the data fill the feature space with sufficient density – again, for a nebulous definition of "sufficient" – the resultant DT should not be overfit.

Given the density and number of training points, we should expect a DT to work rather well.

3.2. *Performance Metrics*

3.2.1. *Accuracy*

The accuracy of a classifier is defined as the ratio of correct predictions to total predictions. I split the data into two parts to calculate it – a training set of around 100 000 points, on which each model was trained, and a testing set of around 50 000 points, on which the predictions were made and compared to the truth.

3.2.2. *Precision*

The precision of a classifier is defined with respect to a given class, and is equivalent to the number of correct positive predictions of that class label divided by the total number of positive predictions of that label. Precision ranges from 0 to 1, with higher scores implying that the classifier has assigned the class label mostly to true members of the class. I used the same training and testing sets as in Section 3.2.1 to calculate precision.

3.2.3. *Recall*

The recall of a classifier is also defined with respect to a given class, and is equivalent to the number of correct positive predictions of that class label divided by the true number of samples with that label. Recall – like precision – ranges from 0 to 1, with higher scores implying that most of the class has been properly labeled by the classifier. I used the same training and testing sets as in Sections 3.2.1 and 3.2.2 to calculate recall.

Classifier	Precision			Recall			Accuracy	Relative Training Time
	Stars	Galaxies	QSOs	Stars	Galaxies	QSOs		
GNB	0.69	0.92	0.87	0.82	0.52	0.83	0.81	1.0
SVM	0.95	0.96	0.99	0.97	0.85	0.99	0.97	26
LDA	0.93	0.91	0.95	0.90	0.83	0.97	0.94	0.35
QDA	0.91	0.94	0.96	0.93	0.84	0.96	0.94	0.28
KNN	0.96	0.93	0.99	0.97	0.87	0.99	0.97	0.21
DT	0.94	0.85	0.98	0.94	0.86	0.98	0.96	1.1

Table 2. The precision, recall, accuracy, and relative training times of the tested classifiers.

3.2.4. *Confusion Matrices*

The confusion matrix of a classifier compares the occurrence of true and predicted labels across all classes. It is often presented in two different ways – once with raw counts, and once with rates such that the sum of all rates for a given true label is 1. I present them both in the same figures, with the counts given by text, and the rates given by color. Either way, an ideal confusion matrix is diagonal, implying that the true and predicted labels for all samples match.

3.2.5. *Learning Curves*

The learning curve of a classifier simply shows the accuracy of that classifier on the training and testing sets as a function of the size of the training set. I tested each classifier five times on ten training sets varying logarithmically in size between 250 and 100 000 samples, and display the mean and variance for each.

3.2.6. *Training Time*

The training time is simply how long it takes to train the classifier. I provide relative training times for all of the classifiers I tested.

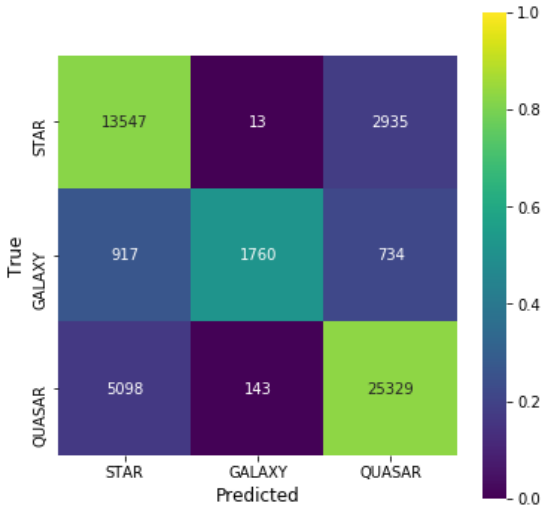
4. ANALYSIS

The accuracy, precision, recall, and relative training times for the classifiers are presented in Table 2. The confusion matrices and learning curves of the classifiers are presented in Figures 3 and 4, respectively.

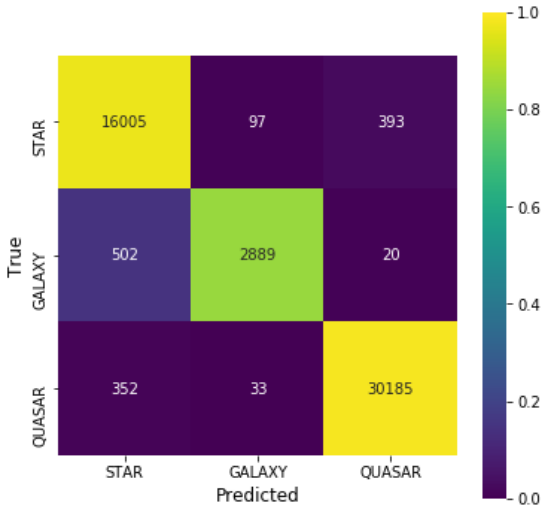
5. SUMMARY

We can see that each classifier performed generally as expected. GNB was the worst at prediction by a large margin, and SVM the slowest by more than an order of magnitude. LDA and QDA performed comparably, suggesting that the data are very nearly linearly separable. The overall best performer, factoring in training time, was KNN, though SVM performed marginally better ignoring training time. SVM does have an advantage over KNN, however, as it can be easily modified to provide the predicted probability of a new sample being in each class.

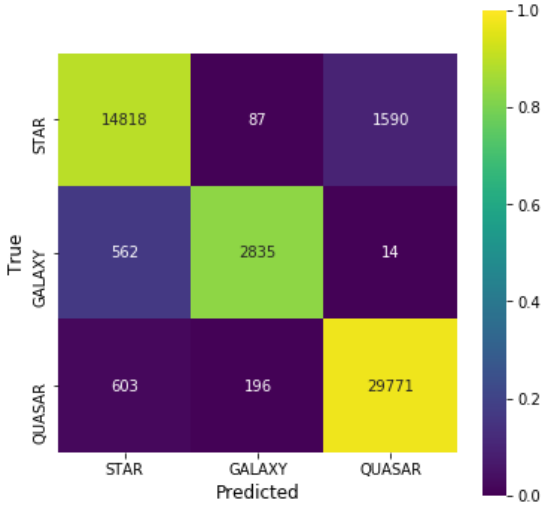
At this point, it should be noted that the data and methods employed here differ from those currently used for the same task by large surveys like SDSS. Rather than



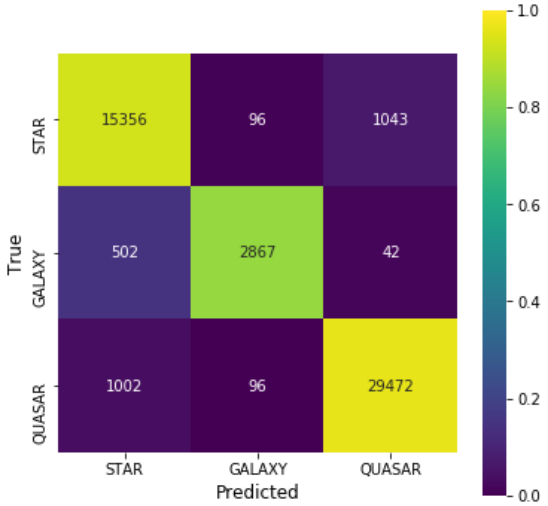
(a) GNB



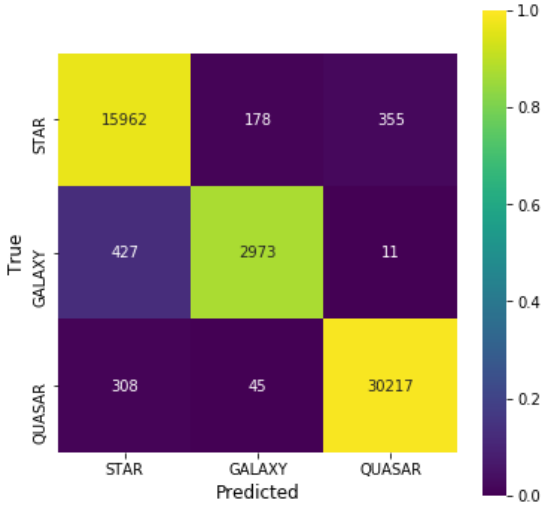
(b) SVM



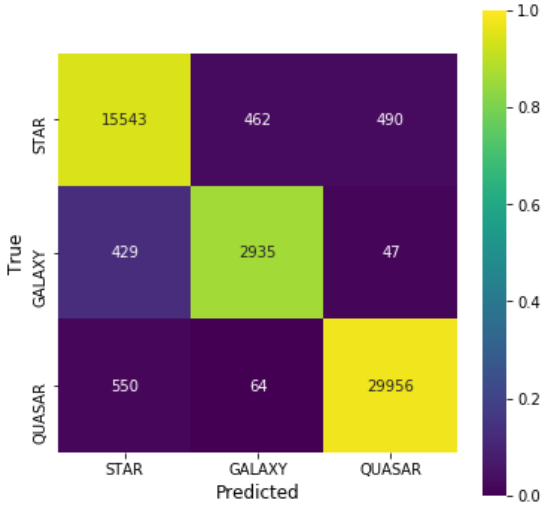
(c) LDA



(d) QDA

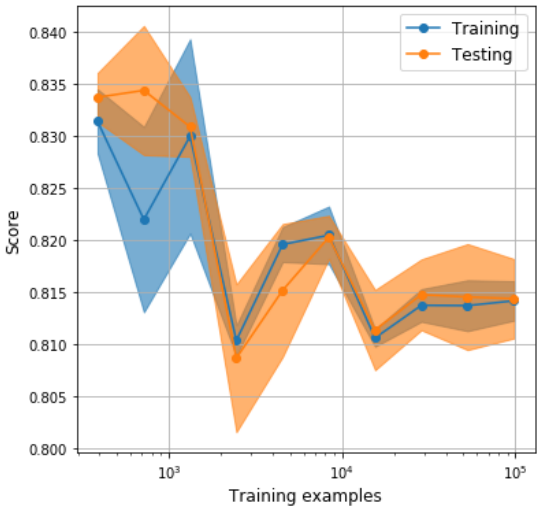


(e) KNN

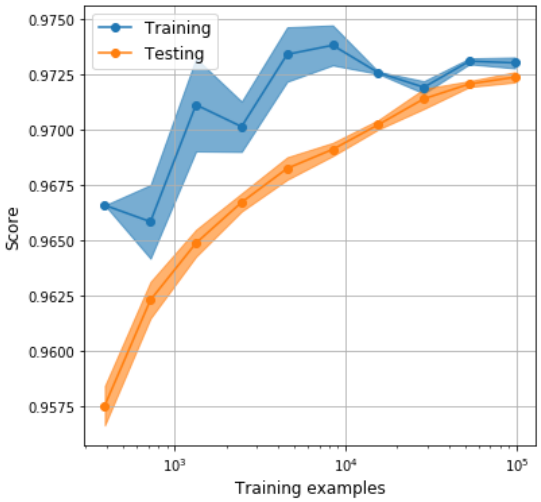


(f) DT

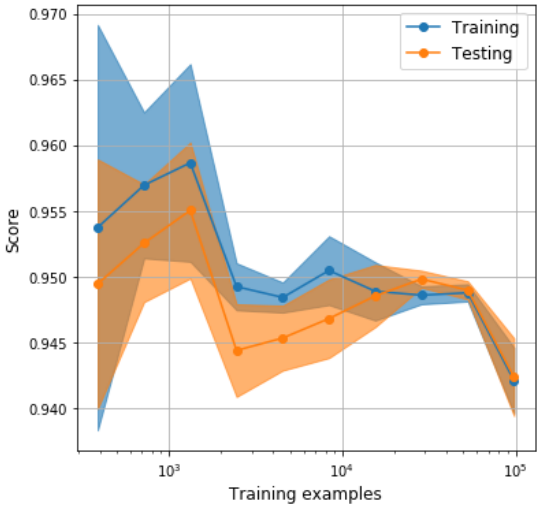
Figure 3. The confusion matrices for each classifier.



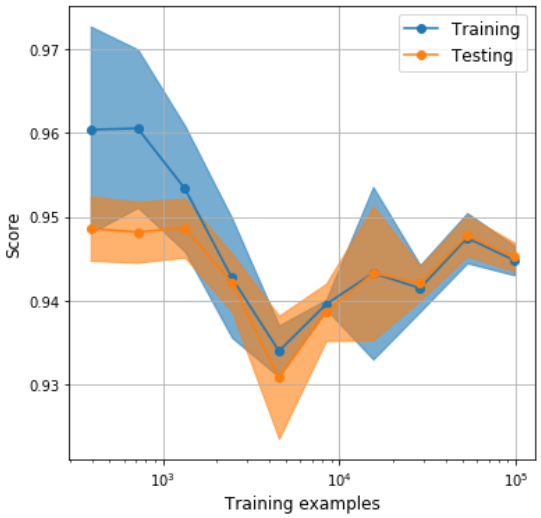
(a) GNB



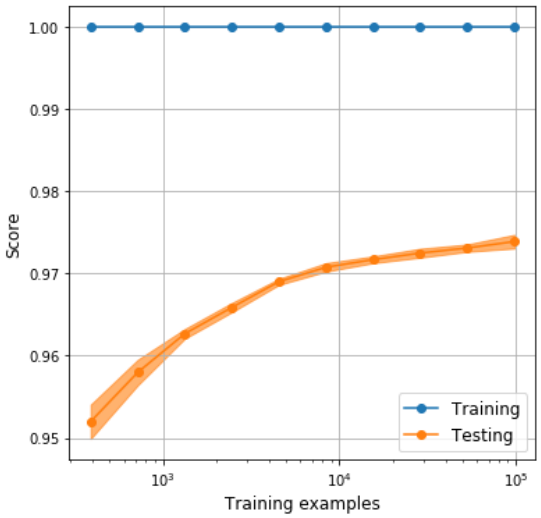
(b) SVM



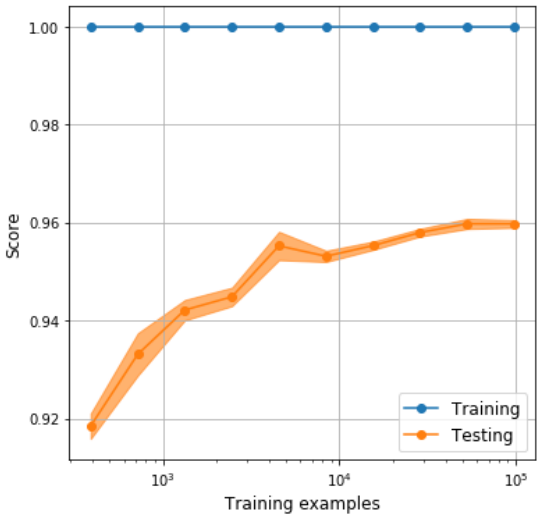
(c) LDA



(d) QDA



(e) KNN



(f) DT

Figure 4. The learning curves for each classifier.

using color indices modern classifiers use whole spectra. One can apply a method called principal component analysis (PCA) to each of the classes of interest to produce a set of prototypical spectra, called eigenspectra. Prediction is then reduced to fitting the eigenspectra of each class to the new point and seeing which fits best. PCA can do much more than classify objects, however, and I recommend reading McGurk, Kimball, and Ivezic (2010). A future project could compare this PCA method to the SVM and KNN methods suggested by this paper.

REFERENCES

Aguado et al. 2018, ApJ, (accepted)

Pedregosa et al. 2011, JMLR 12, 2825–2830

McGurk R., Kimball A., and Ivezic Z. 2010, AJ, 139, 1261