MNRAS 000, 1–3 (2021) Preprint 21 April 2021 Compiled using MNRAS LATEX style file v3.0

Galaxy Image Classification using K-Means Clustering and various Supervised Learning Models

Alex Gass,[1] Michael MacInnis,[1] Samson J. Mercier[1]
[1]*Department of Physics, McGill University, 3600 rue University, Montréal, QC H3A 2T8, Canada*

21 April 2021

ABSTRACT

Since the launch of the Hubble Space Telescope (HST) in 1990, we have been able to capture thousands of incredible images of astronomical objects in our universe. This has resulted in the discovery of a large variety of galaxies. Naturally, astronomers quickly created a system for classifying galaxies. However, as more images were captured, and new types of galaxies were uncovered, classifying galaxies manually became a laborious task. We seek to solve this problem, by utilizing a variety of machine learning techniques to automatically classify galaxies from HST images. In this paper, we use k-means clustering to derive features from each image. The features are then taken as input by a classifier, which will use them to assign the image into one of three categories: a smooth galaxy, a galaxy with features/disks, or a star. We attempted this with five different classifiers and were able to consistently get a lower bound of approximately 76% accuracy, and an upper bound of approximately 80% accuracy.

Key words: Python – Machine Learning – Classification – Galaxies

1 INTRODUCTION

Galaxy classification aims to divide galaxies into different categories based on their morphology. Over the years this field has greatly benefitted from the works of Edwim Hubble and Gérard de Vancouleurs, who defined the various galaxy clusters that we know of today : elliptical, spiral, spiral-barred and many others. However, since the launch of space telescopes such as HST, the number of observed galaxies has exponentially increased (around 100 billion as of today). This meant astronomers could no longer classify galaxies manually, and had to pivot to computational methods.

Before discussing our specific classification problem, we feel it is necessary to give a brief background on the k-means clustering algorithm. k-means clustering is an unsupervised learning approach to machine learning. In unsupervised learning, we are given the feature data, but we are not given the labels they map to. Therefore, we need to find trends within the feature data, to group them together into clusters, in order to classify them. The "k" in k-means represents the number of clusters we are going to group our data into. For each cluster there is a centroid that represents the mean position between all the data points in that specific cluster. The algorithm will continue to assign data points to new clusters based on the nearest centroid to each data point. Then, we recalculate the centroids until there are no new assignments made. In our case, we used k-means clustering to generate centroids from the patch arrays, and from these centroids we were able to obtain our feature data. So k-means clustering was a means for us to obtain our feature data, and from that point we were able to utilize supervised learning with our image labels to classify our data.

We used the data from the Galaxy Zoo Kaggle Challenge (https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge) hosted in 2013-2014 to provide us with our images from the HST. For our raw data, we used the 61,579 images provided in the competition as well as the labels tabulated in a csv file. 70% of these raw images were used to train our various models while the remaining 30% were used to test our models' accuracy. Seeing as our time to work on this project was greatly reduced compared to the candidates of the challenge, we decided to classify our galaxies in 3 classes rather than 37, as outlined in the Decision Tree of the Kaggle challenge.

The three classes we considered are: smooth galaxies, galaxies with features/disks and stars. In order for us to classify our images it was necessary to extract some quantified properties from them. These are known as features in Machine Learning(ML) and we pair each image's features with its label in order to train our supervised learning models. However, before being able to train and test our model we first had to implement an unsupervised learning algorithm in order to generate features from the raw images. Our method for image pre-processing and feature generation were inspired by the ones outlined in Adam Coates and Andrew Ng's paper (1).

2 METHODOLOGY

In this section we will discuss our work pipeline to classify our images.

2.1 Image Pre-Processing

Before trying to classify our images into the three possible categories, we needed to perform pre-processing techniques on the images themselves, so that our feature data obtained from the images

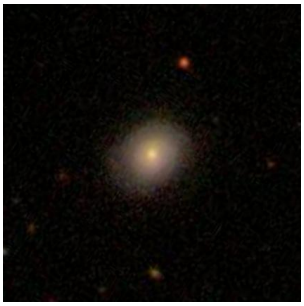

Figure 1. The above figure shows an example image of what our input would be. This is what the image would look like before we have performed any cropping or scaling on it.

Figure 2. The above figure shows a cropped and scaled version of Figure 1. The image is now 16-by-16 pixels.

Figure 3. The above figure demonstrates what one of four patches would look like after cropping and scaling an input image.

would provide better results. In order to do this, we pre-processed the images as follows:

(i) Crop the images from 424-by-424 to 160-by-160 pixels
(ii) Scale the images so they are now 16-by-16 pixels
(iii) Extract 8-by-8 patches from each image, so that each image has 4 patches describing it
(iv) Flatten the patch vectors and combine the three color channels so now the vectors have size 1-by-64
(v) Normalize the patch vectors
(vi) Whiten the patch vectors using ZCA whitening

Each raw image was taken so that the galaxy(or star) would be in the center, which is why we crop and scale our images around their center. Pre-processing also helps reduce the amount of data we have for the next step of our pipeline.

In Figure 1, we can see the image in its original state, Figure 2 shows us the cropped and scaled image, and Figure 3 represents one of four patches of that cropped and scaled image.

After collecting all of our patches from the input images, we then have to flatten, normalize and whiten the patches. We take as input from the patches their pixel intensities, which are arrays with three columns representing the three color channels(R, G, B). After combining the color channels and flattening the arrays so that it is of one dimension for each patch, we then have to normalize and whiten. To normalize, we used the following formula:

$$x^i = \frac{\tilde{x}^i - mean(\tilde{x}^i)}{\sqrt{var(\tilde{x}^i) + 10}} \tag{1}$$

So for each element in each patch array, we normalized the element by subtracting the mean of the array from it, and dividing by the standard deviation. We added a noise factor of 10, to avoid a divide by zero and reduce noise, as was done by Coates and Ng. (1). Normalizing the elements allows for quicker processing, as the pixel intensities are greatly reduced.

The last stage of our image pre-processing consisted of whitening our patch arrays. We needed to whiten our patches because k-means clustering often generates centroids that are highly correlated to each other, while we want centroids that span the data more evenly. Therefore, if we didn't use whitening, we would get many clusters with no data points in them, leading to many centroids with values of zero. We used ZCA whitening, as it is a common choice for whitening, and our whitened points were calculated as:

$$x_{new} = V(D + \epsilon_{zca} I)^{-\frac{1}{2}} V^T x \tag{2}$$

Where V is the whitening matrix, D represents our dictionary of centroid values, and ϵ_{zca} is a small constant, we used 0.1. This process is further detailed in Coates and Ng.'s paper (1).

2.2 Feature Generation

From the image pre-processing we get a 1-by-64 vector for each patch. We run a Mini Batch k-means algorithm on our patch vectors with k=500 centroids, a batch size of 1000 and we take a number of iterations sufficient for our algorithm to converge, 20-30.

From this k-means algorithm we get k centroid vectors, all of size 1-by-64. Putting these centroids in a dictionary $\mathcal{D}$ gives us a bank of filters with $\mathcal{D} \in \mathcal{R}^{k \times 64}$. We can then use $\mathcal{D}$ to create a function f that will help map all our 1-by-64 patch vectors to 1-by-k feature vectors. In our case, we chose f to be a matrix multiplication function:

$$f(x) = \mathcal{D} * x \tag{3}$$

$x \in \mathcal{R}^{64}$ is our patch vector and $f(x) = y \in \mathcal{R}^k$ is our feature vector. Applying this function to all our patches allows us to find features for all our images.

2.3 Supervised Learning for Classification

At this stage, we have our feature vectors for each image, as well as the labels from the solutions file, so we are ready to begin training our model with various supervised learning techniques. We tried multiple supervised learning models for multi-classification (as we have three possible labels to map to), and in the next section we'll discuss our results, as well as the models that gave us the best accuracies.

RMS Accuracies (%)	Model
79.6	SVC
77.2	Logistic Regression
77.3	Random Forest Classifier
77.5	Gradient Boosting Classifier
76.1	Radius Neighbors Classifier

Table 1. The above table displays the different accuracies obtained from running different supervised learning algorithms.

2.4 Testing Accuracy

We used the Root Mean Squared Error as our evaluation metric for our various supervised learning methods. The accuracy was calculated following the below equation:

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2} \tag{4}$$

Where N is the product of the number of images and the number of classes, p_i is our predicted value for each image's label and a_i is the actual value. We used the root mean squared function to test our accuracy, as the labels we were given were probabilities of each class. Therefore, to train our models, we took the highest probability category for each image, and used that as our labels.

When testing our models, we had them predict the labels as probabilities. Therefore, our models would predict the probability that each image belongs to each category, and the root mean squared formula would be calculated to compare the probabilities we outputted to the probabilities given in the solutions file.

3 ACCURACY FOR VARIOUS MODELS

In table 1 we show the accuracies obtained from using different supervised learning algorithms.

We trained on 14,000 images and tested on 6,000 images. As can be seen from the table, the one vs rest classifier used with a support vector classifier gave us our best results. In comparison, the winners of the Kaggle challenge got approximately 93% accuracy from their results. There are a couple potential reasons for the drop in accuracy in our implementation compared to theirs.

Firstly, we could not train our model on more than 14,000 images. This was because our kernels would crash when trying to input more images. The literature we are comparing to trained on approximately 61,000 images, which would cause the accuracy to increase, with that much more data. Especially when we are working in higher dimensions, k-means needs much more data to produce sufficient accuracies.

Another potential reason for the drop in accuracy between our two implementations are the parameters we are using for batch size, number of centroids, epsilon parameter when whitening, and the noise parameter when normalizing. We followed the literature when setting these parameters, and have tried some testing with different values. It is possible though, that with our implementation, there is a different set of parameters that would give us a better accuracy. More testing would have to be done to see if this is the case.

3.1 Next Steps

We believe our k-means feature generator was what prevented us from attaining a higher accuracy. In the future, we think using a different clustering technique could improve the accuracy, as k-means does not produce as sufficient results on smaller amounts of data.

We also hope to complete more testing on our models, to possibly improve the accuracy by changing parameters. One potential next step is creating a script that can continuously run, while initializing random values for the different parameters. And for each run, we record the parameters, as well as the accuracy, to find the our optimal parameters that produce the highest accuracy.

4 CONCLUSIONS

From our results, we have found that we can classify our images into the three categories of a smooth galaxy, galaxies with features/disks, and stars, with approximately 80% accuracy. We were approximately 13% off from the literature value, but also trained our model on only approximately 1/4 of the data. Overall, we found that using k-means clustering for feature generation, and trying different supervised learning algorithms produced sufficient results.

REFERENCES

Coates A., Ng A.Y. (2012) Learning Feature Representations with K-Means. In: Montavon G., Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 7700. Springer, Berlin, Heidelberg.

ACKNOWLEDGEMENTS

We would like to thank Professor Adrian Liu for providing us with constructive feedback and also for providing the raw data from Kaggle.

This paper has been typeset from a TEX/LATEX file prepared by the author.