Draft version December 17, 2022
Typeset using LaTeX default style in AASTeX631

Astro101 Final Project: (Un)SUpervised Clustering Of Variable Stellar Objects ((U)SUCOVSO)

Oscar Scholin[1] and Graham Hirsch[1]

[1] *Pomona College*
170 E Sixth Street
Claremont, 91711, CA USA

ABSTRACT

We present a project using the ASAS-SN catalog (Christy et al. (2022a)) [a)] as the data of focus for supervised and unsupervised neural networks we implement to classify variable stars. We outline a plan to use Marsh et al. (2017) to identify unique subclusters within the contact binary systems. We give an example neural network using the famous iris dataset (Kaggle (2017)) as an illustration of our technique; we also describe the theoretical workings of three unsupervised algorithms—K-Means, OPTICS, and DBSCAN. We define a list of 36 variability indices—included with formulae and comments in the Appendix—to quantify lightcurves, which serves as the actual input data for the networks. Our supervised model has $> 85\%$ accuracy when run on unseen ASAS-SN data, illustrating that our variability indices do a reasonably good job of discriminating variability type. Our unsupervised attempts, however, generate no usable clusters, but reveal some problems with our data preparation and inform future directions for the project. All the code we have written, along with our data products, is freely available and documented on GitHub. [b)]

1. INTRODUCTION

The central idea behind this project is to use a deep learning neural network (IBM (2022)), which is a type of algorithm that take as input training data, which can be purely numerical or image-based, in order to output a prediction of what category of variable object the input object belongs. There are two main types of neural networks: supervised and unsupervised.

A supervised network means we have labels for the training data—that is, what class a particular input vector corresponds to. An unsupervised model does not include any labels. So we do not know a priori what the data should be (e.g., a cat or a dog); rather, the computer "clusters" the data spatially in various ways to determine what the labels should be. Therefore, utilizing unsupervised clustering effectively requires more finesse to generate meaningful assignments of data.

(a) Virginica, Farm (2022a)

(b) Versicolor, Flora (2022)

(c) Setosa, Farm (2022b)

Figure 1: Images of three iris species used in the example neural network.

[a)] https://asas-sn.osu.edu/variables/

[b)] https://github.com/oscars47/Astro101-Final-Project

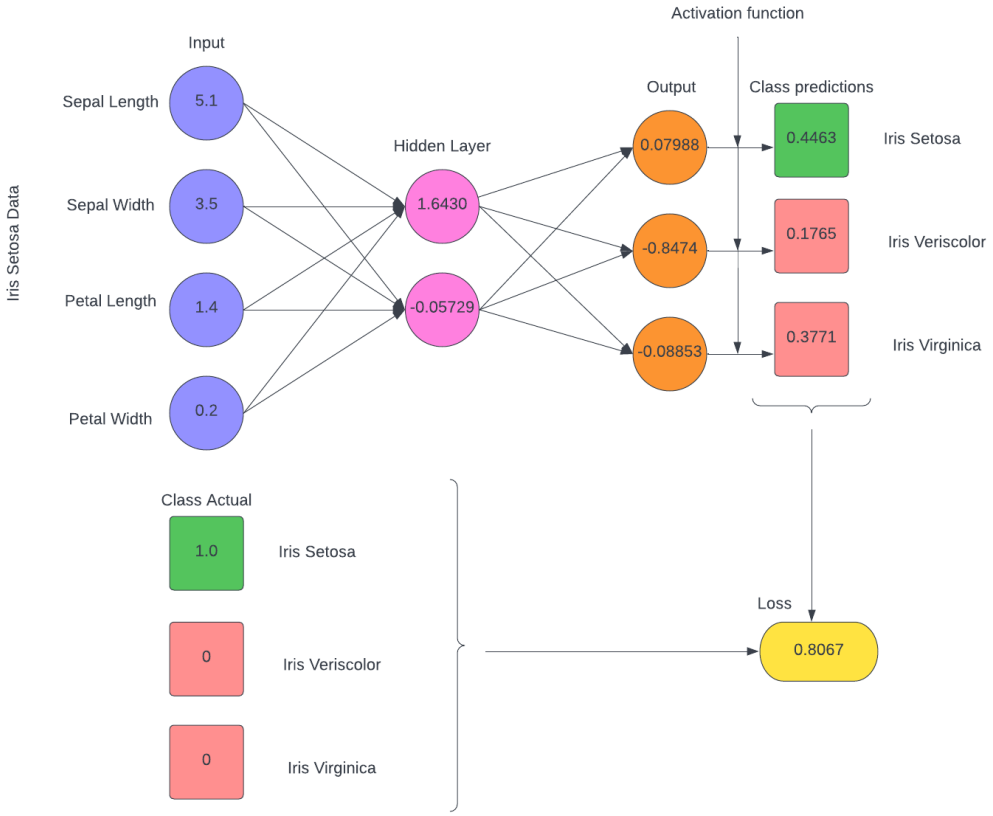


Figure 2: Sample neural network for one entry in the Iris dataset Kaggle (2017). To read the diagram, follow the arrows. The input is data values, which undergo a linear transformation as described in the Introduction and become the output of the single hidden layer. The outputs undergo a similar linear transformation and then must be converted via an activation function into values in $[0, 1]$ to be interpreted as a probability. Loss is computed by comparing predicted class results with class targets.

The first focus of the project was implementing a supervised network on numerical data generated from the lightcurves of every object in the ASAS-SN dataset in order to classify the type of variable object given 19 choices (see figure 9a; Christy et al. (2022a)). We also explore using an unsupervised model to classify the same data as a proof of concept for analyzing the lightcurves of contact binary systems (Marsh et al. (2017)) to tease apart variations that might indicate something about the makeup and evolutionary history of the objects. In the Methodology we give an overview of an example neural network we prepared of a simple example studying irises as well as explain the three different unsupervised methods we tried. We then present our Results for both approaches and offer an analysis in the Discussion, as well as suggestions for future improvements.

2. METHODOLOGY

In order to illustrate how a neural network works, consider figure 2 [1]. The problem modeled here is how to distinguish between three types of irises: Virginica, Versicolor, and Setosa (see figure 1). You can read this diagram by following the flow of the arrows, and imagine a Pachinko ball passing along the path from input to output. [2]

We start with our data, which here is from Kaggle (2017) and consists of four parameters—sepal length, sepal width, petal length, and petal width—for each flower. So we can consider a single input vector for a single flower as the purple circles in the figure with each the values listed in the center. Then we can image drawing lines from each of these inputs to a "hidden layer" with two neurons shown in orange. This is the backbone of the network, which often involve

[1] You can follow along at https://github.com/oscars47/Astro101-Final-Project/blob/main/nn_sample/Iris_forward.ipynb

[2] Thanks to Prof. Choi for this great metaphor!

many hidden layers with upwards of 200 neurons. Finding the optimal values for these "hyperparameters", i.e. those parameters for the model architecture that the user must determine before initializing training, is not straightforward and will be discussed in the Methodology. For now though, we can use this simplified version of one hidden layer with two neurons. Each of the lines connecting the input and hidden layer have a "weight" $\vec{w}_i$, a vector of numerical values corresponding to each of the neurons in the hidden layer. Moreover, each hidden layer has an associated "bias" b, also a numerical value. These are not shown for visual clarity. The values for each of the hidden layers is given by

$$\hat{y}_i = \vec{x} \cdot \vec{w}_i + b \tag{1}$$

where x is the input layer and $\cdot$ is the dot product (Dhingra (2021)). If we randomly assign the weights and biases, we can perform the calculations as shown in figure 3.

```python
# time for hidden layer!
# we want 5 neurons, so we will have a list of length 2 containins 4 values each for the weights
# and then a list with 2 elements for biases

# choose randomly: initialize weights!
weights = []
for i in range(2):
    neuron_list = []
    for i in range(4):
        neuron_list.append((np.random.random())*2 -1)
    weights.append(neuron_list)

# now for biases
biases=[]
for i in range(2):
    biases.append(((np.random.random())*5))

print('weights1:', np.array(weights))
print('-------')
print('biases1:', np.array(biases))
print('-------')

# now pass the data through the network
out = np.dot(weights, input) + biases
print('out1:', np.array(out))
```

(a) Code snippet showing hidden layer definition. The first part initializes random weights and biases and then prints them, and the final part takes the dot product as in equation 1.

```
weights1: [[-0.20110904 -0.16360342  0.72501694 -0.39025517]
 [-0.13413757 -0.59175585 -0.58734769  0.5912798 ]]
-------
biases1: [2.30429604 3.40198756]
-------
out1: [ 1.64300063 -0.05729033]
```

(b) Values for the weights, the biases, and output for the hidden layer from the code in figure 3a. For the weights, each element in the overall list corresponds to a particular neuron, and each element within the sublist corresponds to a unique input value.

Figure 3: Hidden layer results for iris network.

Then we simply do the same trick again to connect the hidden layer to the output layer. Since we want to predict the class of each flower, which has three possibilities, we have three orange output layers. You may notice that some of the outputs have negative values. Since we want to ultimately choose the predicted class based on a probability, we need a way of converting these output values to numbers between $[0, 1]$. This is why I have labeled "Activation function" intersecting the arrows from the Output layer. An Activation function simply maps an input to a value between 0 and 1. The function we have chosen is called Softmax (Wood (2019)), as shown in figure 4.

Now, on the far right of figure 2, we have in squares our class prediction layer. You can see the values listed for each of the classes, and the highest value is ≈ 0.45, which corresponds to Setosa. We can compare this prediction to the actual, which is a three dimensional vector with a 1 in Setosa and a 0 everywhere else. We can compute the "loss" of the model as a means of quantifying how wrong the model is. The loss function I have used in this example is called Categorical Cross Entropy (Kumar (2020)), described in figure 5.

The central idea behind a neural network is figuring out how to tweak the values for the weights and biases in order to predict the optimal outcome. Neural networks do this through backpropogation (Al-Masri (2022)), a form of gradient optimization: that is, to find the partial derivatives of the loss function with respect to each weight and bias to measure how changing a particular term most affects the overall loss—which is essentially just a very high dimension calculus optimization problem (there is one weight for every input-output connection and one bias per layer, so here there are 15 parameters to optimize); this is why large neural networks with large input vectors can take up a great deal of computing resources.

This is the essence of a supervised neural network. We can extend this logic to considering processing not just one vector but a "batch", which enables the process to accelerate. In this case, the input X is not a vector but a matrix,

```python
# now need to pass through activation function to get values bn 0 and 1
# here use softmax
exp_vals = np.exp(out2 - np.max(out2)) #subtract max per row
probabilites = exp_vals / np.sum(exp_vals)
print(probabilites)
print('------')
print('chosen class:', species_list[np.argmax(probabilites)])
```

```
[0.44630219 0.17657212 0.37712569]
------
chosen class: Iris-setosa
```

(a) Code snippet showing activation function computation corresponding to the equation to the right.

$$\sigma(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Figure 4: Softmax activation function and results. Note in the equation $\vec{x}$ is the input vector, so x_i is ith value in the input vector. The denominator ensures we have a normalized probability.

```python
# need to compute loss
# use categorical cross entropy loss
y_pred_clipped = np.clip(probabilites, 1e-7, 1-1e-7) #need to clip close to 0
correct_confidences = np.sum(y_pred_clipped*target)
negative_log_likelihoods = -np.log(correct_confidences)
print(negative_log_likelihoods)
```

```
0.8067589938167536
```

(a) Code snippet showing loss computation corresponding to the equation to the right. The chosen class corresponds to the index of the largest probability (hence the "np.argmax()").

$$\text{Loss} = -\sum_i L_i \log(S_i)$$

Figure 5: Hidden layer results for iris network. Note in the equation that L_i is the ith output from the actual/target vector and S_i is the ith value in the prediction vector.

and so is W_i, so our equation for the vector of neuron values now becomes the matrix product:

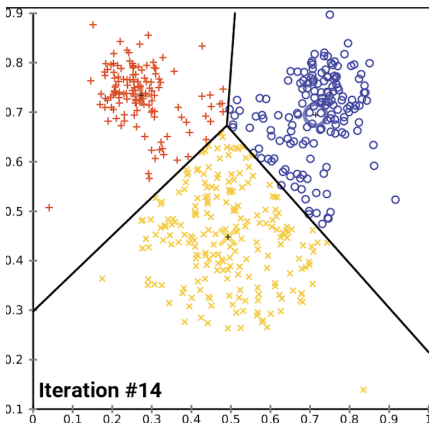
$$\hat{\vec{y}} = XW_i^{\intercal} + b \tag{2}$$

But the theory of the training remains the same. We should also note that when preparing data to train the model, it is imperative to create a training/validation split of the data (Goyal (2021)): there is the danger of *overfitting*, in which the model preferentially learns the relationships between the its input data but cannot be generalized to other similar types of data.

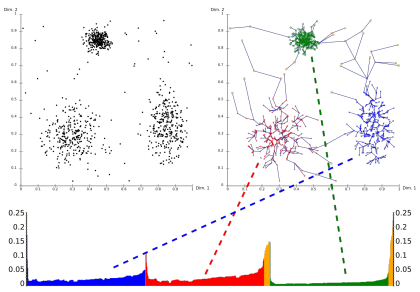
Unsupervised training is a bit trickier, as again we have no way of automatically validating the results at each step. We implemented three algorithms, as shown in figure 6: K-Means, OPTICS, DBSCAN.

K-Means randomly assigns cluster centroids (note: we can either specify beforehand the number of clusters, or create a plot of Within-Cluster-Sum-of-Squares (WCSS) plot as shown in figure 16b and determine the "elbow" or inflection point of the curve, which corresponds to the number of clusters). Each time the model iterates it recalculates the Within-Cluster variance and tries to minimize this quantity by adjusting the location of the centroids and boundaries of the classes. Eventually it converges, usually to spherically-symmetrical clusters (Anwla (2021)).

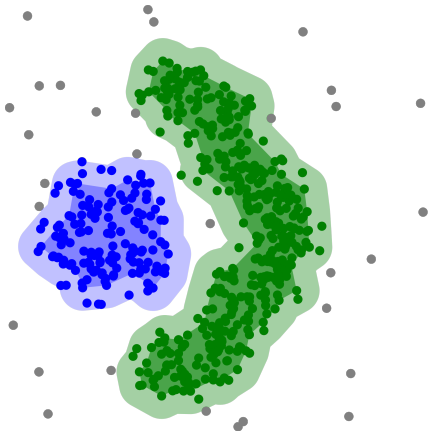
However, since we would not necessarily expect out data to be represented by symmetric clusters, we investigated two other methods: OPTICS (Ordering Points to Identify Cluster Structure) and DBSCAN (Density-Based Spatial Clustering of Applications with Noise). OPTICS takes as a hyperparameter only the number of minimum points $\min_{\text{pts}}$, the number of minimum points to form a cluster. DBSCAN is similar, but also requires the parameter ϵ which is roughly the distance between points within the same cluster, which we can determine by plotting the k-Nearest Neighbors (kNN) and finding the elbow of the curve as with the WCSS plot (see figure 16a). DBSCAN is better than K-Means at non-linear clustering, but OPTICS has the advantage over DBSCAN as it uses a nonuniform ϵ when defining neighborhoods of points, so it provides another layer of complexity. DBSCAN randomly chooses a point; if there are $\geq \min_{\text{pts}}$ points within a distance of ϵ, then this point is marked as a centroid and the model begins to form clusters; if there are insufficient points, then it is classified as an outlier. If we the other points within the cluster have at $\geq \min_{\text{pts}}$, then we add these other points to the original cluster. We repeat this process until we have considered



(a) Ilustration of K-Means algorithm, which performs an iterative process of assignig a "center of mass" for each cluster and adjusting until losses are minimized. Image credit: Wikipedia (2022a)



(b) Illustration of OPTICS algorithm, showing how steep increases in reachilbility lead to the distinction between clusters. Image credit: Wikipedia (2022b)



(c) Illustration of DBSCAN algorithm, which can identify non-uniform clusters and well as outliers using a distance parameter and minimum neighbors. Image credit: Wikipedia (2022c)

Figure 6: Three unsupervised clustering algorithms employed.

all points (Yıldırım (2020)). The difference in OPTICS is we compute, for each point, the "reachibility distance" of the $\min_{pts}$ closest points. Figure 6b illustrates how we can continue to calculate reachilibility distances to sucessive points, and sharp increases in this measure indicate the end of that particular cluster (Sinclair (2019)).

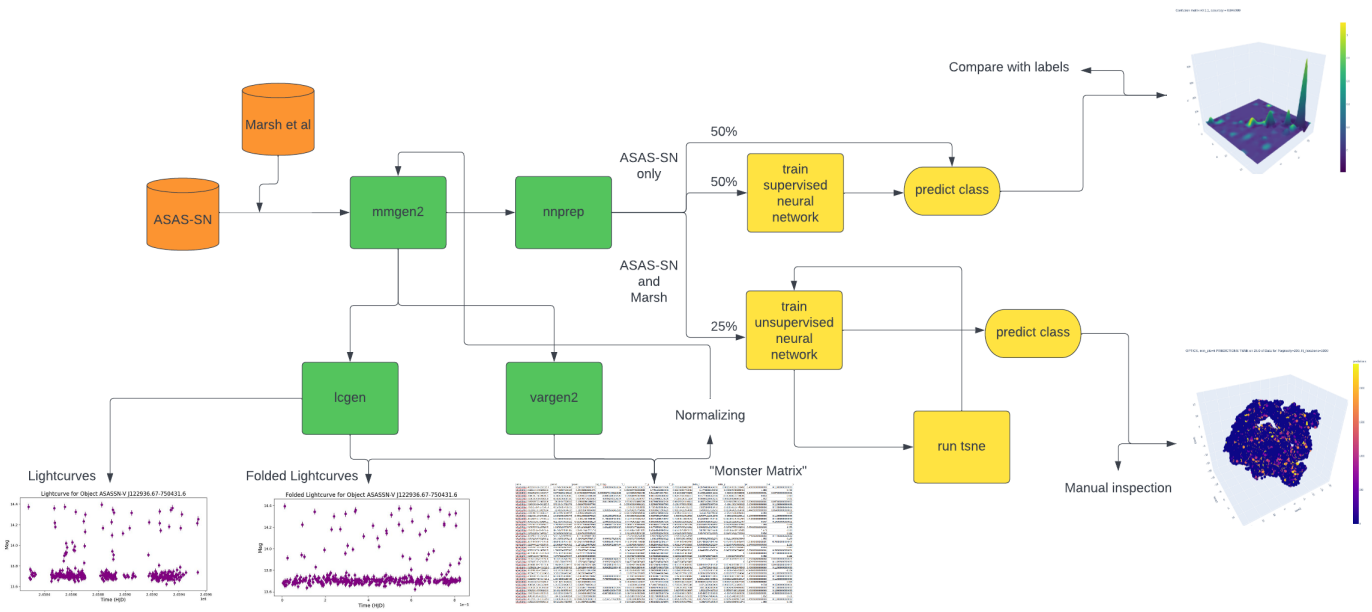


Figure 7: A map of the flow of data through code for this project. Data (orange) initially is fed into a data processing pipeline (green) to quantify specific features about a lightcurve, which is then split up two main channels: for use in training a supervised neural network and an unsupervised network, shown in yellow. The output is data representing the class or similarity/asimilarity of input objects.

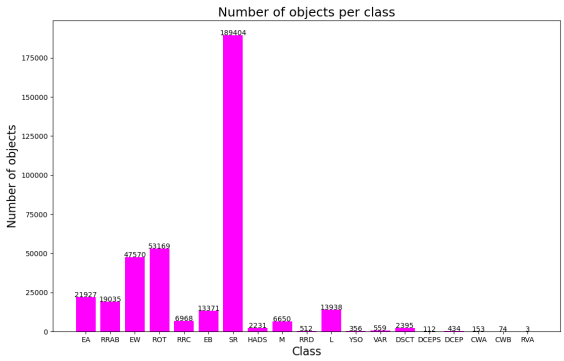
Now that we have established the theoretical basis for the project, we can more concretely describe the actual process we underwent, which figure 7 summarizes. We start with data from ASAS-SN (Jayasinghe et al. (2020), Christy et al.

(2022a)) and (for contact binaries) Marsh et al. (2017) in the form of detailed time-magnitude data for each object, which is fed into the masterdataprep pipeline, which allows us to visualize raw lightcurves as well as to compute 36 distinct variability indices adapting Pashchenko et al. (2018), Sokolovsky et al. (2017), Lopes et al. (2015), Jayasinghe et al. (2020), Christy et al. (2022a), and Clarke (2002), described in tables 1, 2, 3 along with descriptions interpreting each measure. This is really the heart of the project: we need a way to quantify lightcurves so the computer can analyze them. The output for this pipeline is a CSV we have termed "Monster Matrix", which stores all of these parameters for each object.

In Python, we prepared the data for training by converting our Pandas dataframes into Numpy arrays. For supervised training, we assigned one-hot encoded target vectors (a vector of 19 0s with a 1 corresponding to the index of the class of variable objects listed in figure 9a. Note: Figure 9b shows the frequency of different classes of objects in the catalog). We then split up the data into 50% for training, which would be split again into 80% and 20% for training and per epoch, or iteration, validation. This split here is important because the model is actually optimizing the validation loss, not the training loss, for better results. The remaining 50% is used as independent validation after the model is trained to verify accuracy. We note that we used the Python packages Tensorflow and its derivative Keras to implement our networks, and again the code is available via the GitHub link in the Abstract.

RF Classification	Description	N_{Known}	N_{New}	N_{New}/N_{Known}
CWA	W Virginis type variables with P > 8 d	153	-	-
CWB	W Virginis type variables with P < 8 d	73	1	0.01
DCEP	δ Cephei-type classical Cepheid variables	432	2	<0.01
DCEPS	First overtone Cepheid variables	109	3	0.03
DSCT	δ Scuti type variables	848	1547	1.82
EA	Detached Algol-type binaries	17447	4480	0.26
EB	β Lyrae-type binaries	11820	1551	0.13
EW	W Ursae Majoris type binaries	42737	4833	0.11
GCAS	γ Cassiopeiae variables	-	-	-
HADS	High amplitude δ Scuti type variables	1725	506	0.29
L	Irregular Variables	9152	4786	0.52
M	Mira variables	6287	363	0.06
ROT	Spotted Variables with rotational modulation	14755	38414	2.60
RRAB	Fundamental Mode RR Lyrae variables	18455	580	0.03
RRC	First Overtone RR Lyrae variables	6518	450	0.07
RRD	Double Mode RR Lyrae variables	482	30	0.06
RVA	RV Tauri variables (Subtype A)	3	-	-
SR	Semi-regular variables	131479	57925	0.44
YSO	Young Stellar Objects	209	147	0.70
VAR	Variable star of unspecified type	150	409	2.73
Total		262834	116027	0.44

(a) Key to variable names and their abbreviations.



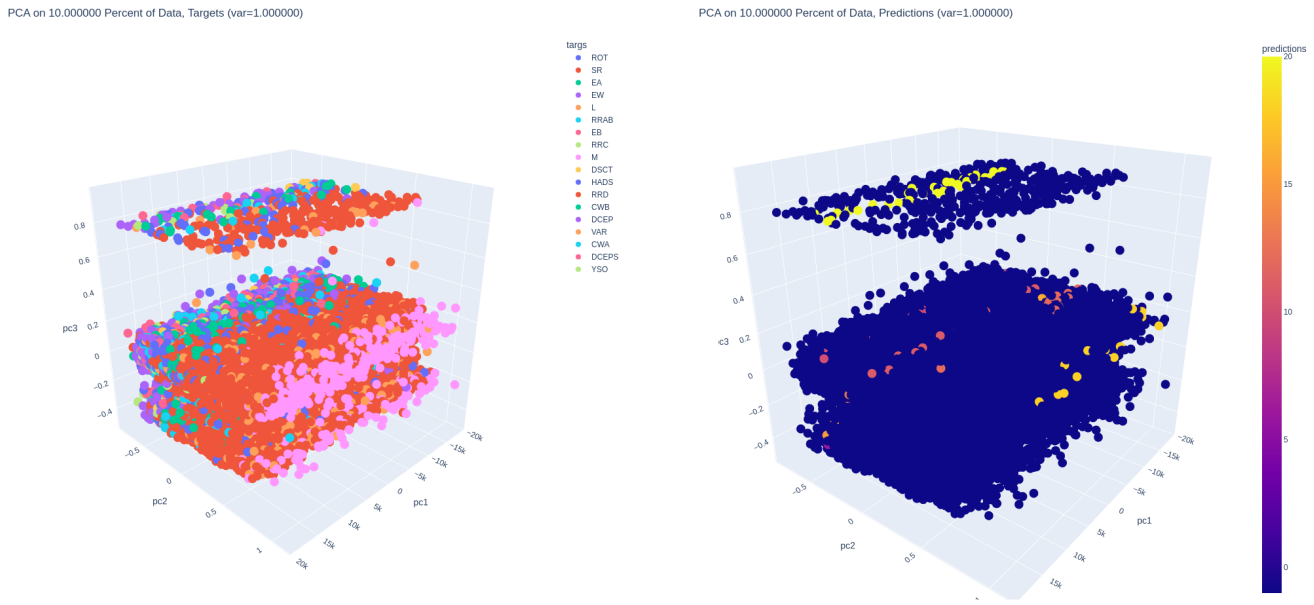
(b) Plot of the distribution of variable types within the dataset. Note the exceptionally high number of SR.

Figure 8: ASAS-SN information, using Christy et al. (2022a).

For unsupervised clustering, we used 25% of the data and calculated their t-SNE representations. Data dimension reduction is a common technique (e.g., Melit Devassy et al. (2020)) in order to simplify the problem for the computer; however, this can introduce problems of its own which we go more in depth in the Discussion section. t-SNE is a non-linear dimensionality reduction technique (SKLearn (2022a)), which we used as an alternative to the linear Principal Component Analysis (PCA) (SKLearn (2022b). We used t-SNE instead of PCA because the PCA compressions, although it had a high capture of variance (> 99%), it returned, when visualized, a plot with three distinct layers irrespective of the boundaries of variable type (see figure 9). Figure 10 illustrates one of the two hyperparameters used in the t-SNE algorithm: perplexity and $N_{\text{iterations}}$. These three plots illustrate how determining, for example, the right value of $N_{\text{iterations}}$ is not entirely obvious. What we must do is experiment by producing a variety of representations, plotting them, and picking out what perplexity and $N_{\text{iterations}}$ values give us the best results. (While $N_{\text{iterations}}$ is more straightforward as the number of times we run the algorithm, perplexity may seem a bit more unintuitive, but all it is a measure of whether to weight local or global features as more important to preserve.) However this is also complicated by the fact that topological features that emerge may be noise, as t-SNE is a probabilistic algorithm. Again, the only solution here is to experiment and re-run certain combinations of parameters.

3. RESULTS

In figures 11 and 12, we present the results of the supervised clustering for three groups of indices, as in tables 1, 2, 3. Figures 13 and 14 show the t-SNE plots for all and periodic only variables; figure 15 shows the results for K-



(a) PCA representation of 10% of data colored by actual class.

(b) PCA representation of 10% of data colored by predicted class from OPTICS, which identified 1736 unique clusters.

Figure 9: Initial PCA efforts.

(a) Sample t-SNE at 0 iterations.

(b) Sample t-SNE at 100 iterations.

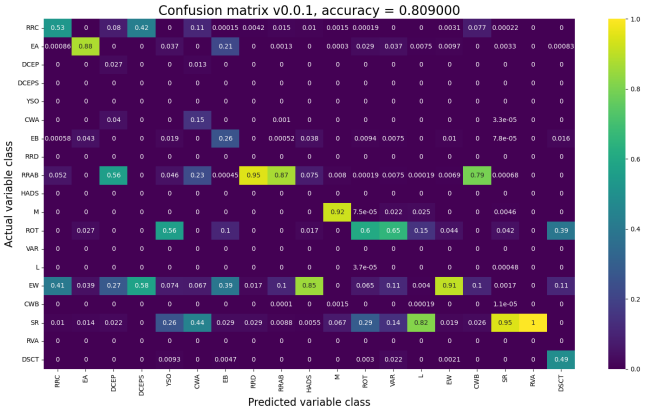
(c) Sample t-SNE at 2000 iterations.

Figure 10: Evolution of the t-SNE algorithm, using perplexity 10. From Wattenberg et al. (2016).

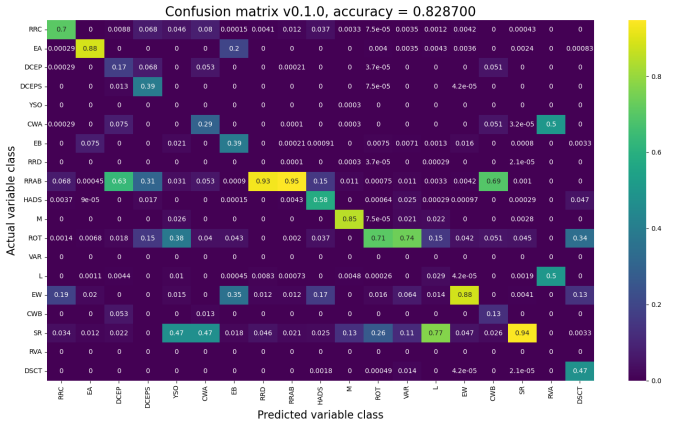
Means, OPTICS, and DBSCAN. Figure 17 shows the t-SNE representations for the RR-Lyrae variables to investigate a subclass of a variable type.

4. DISCUSSION

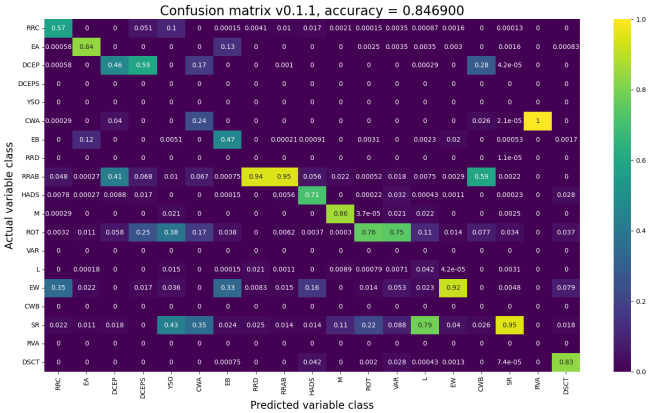
For the supervised results on the untouched 50% of the initial data we split, in figure 11 we notice a clear increase in accuracy as we go from Group 1 to Group 2 to Group 3; interestingly, the change from Group 2 to Group 3 is more significant from Group 1 to Group 2—in Group 3 we added period and power generated from the Lomb-Scargle periodogram (which we will come back to). Higher values along the diagonal indicate a higher percentage of correctly predicting the actual class; off-diagonal values indicate misclassifications. Figure 12 contextualizes these results by giving us a third dimension to illustrate the number of objects for each combination. This plot indicates how for some



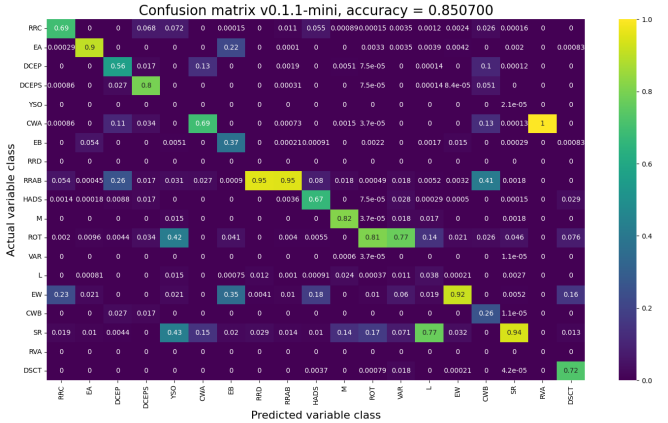
(a) Results using Group 1, 5 hidden layers.



(b) Results using Group 2, 5 hidden layers.



(c) Results using Group 3, 5 hidden layers.

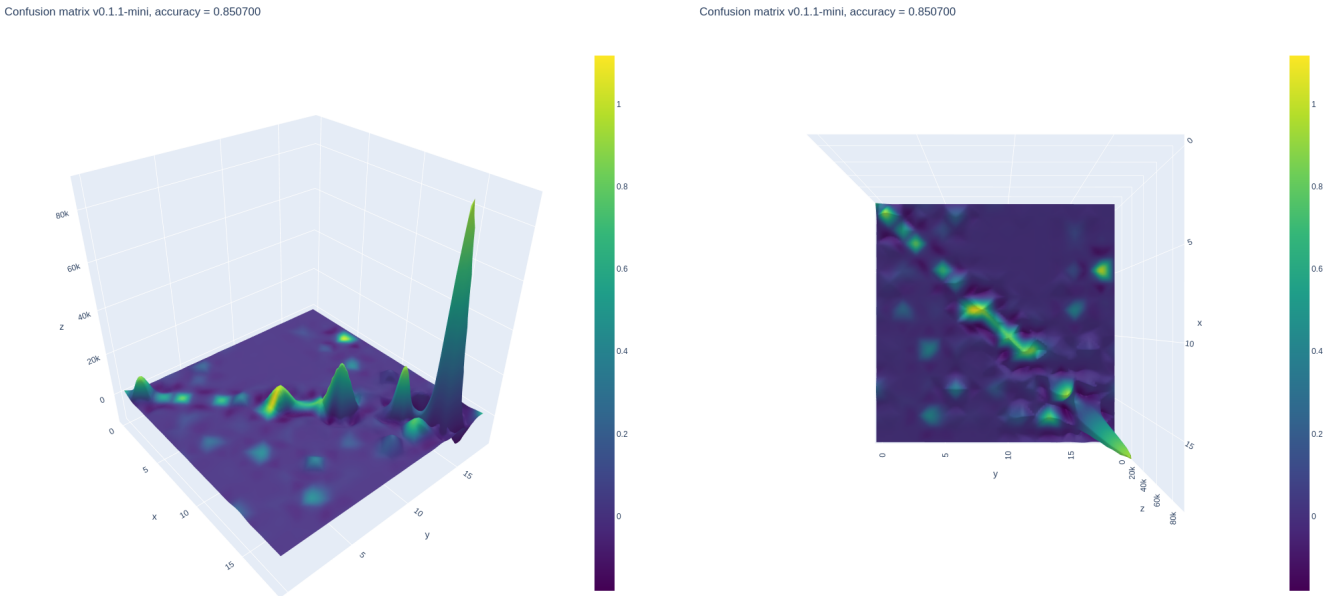


(d) Results using Group 3, 1 hidden layer.

Figure 11: Confusion matrices for four iterations of the supervised neural network fed the untouched 50% of the data. To read these plots, note that the predicted class is on the x-axis and the actual class is on the y, so the diagonal elements of the matrix represent percent of the time the model correctly classified that particular type of class; the off diagonal elements represent false classifications. Note the increase of accuracy from figures 11a to 11d, in particular the increase from 11c to 11d, which illustrates that perhaps counterintuitively the model with fewer hidden layers performed better than that with 5.

objects, e.g., SR, there are many objects in the training and validation data, so the model is more likely to be able to discriminate these objects, whereas for other objects, e.g. RVA, there are a very few number so the model will likely not predict these well. Also, some of the objects are within the same class, e.g., RRD vs RRAB, so it makes sense that the model might confuse these. Further exploring improving the model's predictive power for these indices will help us in our ultimate endeavor of clustering contact binaries, i.e. identifying sub-classes within a particular cluster. A surprising result is that, as the increase from 11c to 11d shows, the model with fewer hidden layers (only one) performed better than that with 5. This is suprising because one would expect that more hidden layers reflects a greater complexity of model and thus a greater predictive power, but these results seem to contradict that. One possibility is that we got really lucky in training the one hidden layer model and, since we only ran each model for 100 different unique combinations of hyperparameters and then optimized all of the weights and biases given those 100 different network configurations, if we were able to sample a much wider variety and train for even longer, the more complicated network might win in the end. Nevertheless this is a notable result.

The t-SNE clustering in figure 13 is a bit of a hot mess—as we get to perpelxity values of roughly 200, we get a bit better distinctions, but the plots mostly look like a blobbing mix of various colors hence classes. An interesting feature that is preserved is a distinction between red and orange (L and SR), which represent aperiodic variables, and



(a) Side view.

(b) Top-down view.

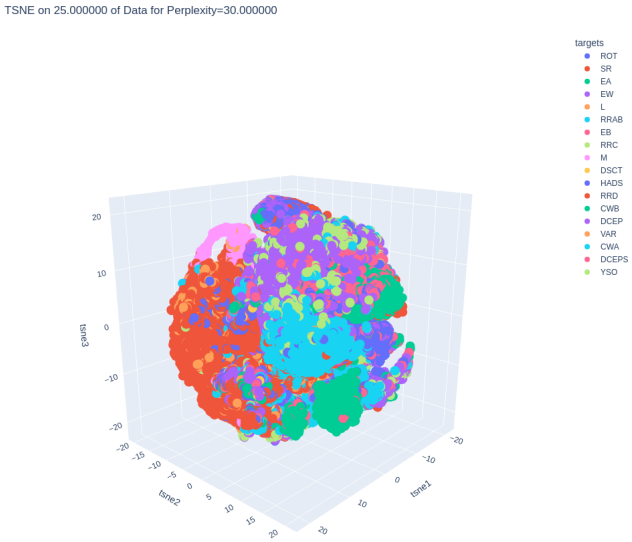
Figure 12: Results equivalent to those in 11d, but here plotted in 3D with the height representing the number of objects (note the huge peak for SR, as noted in 9b).

the rest, which are periodic (except for the pink M, which seem to be attached to the aperiodic blob). To further investigate this, we ran t-SNE only on periodic variables, as shown in figure 14. While we were unable to achieve significant discrete clusters, for a perplexity value of 200 there was a balance between a single blob and more fractures. An interesting feature preserved across all variations in the periodic case was the pink "M island".

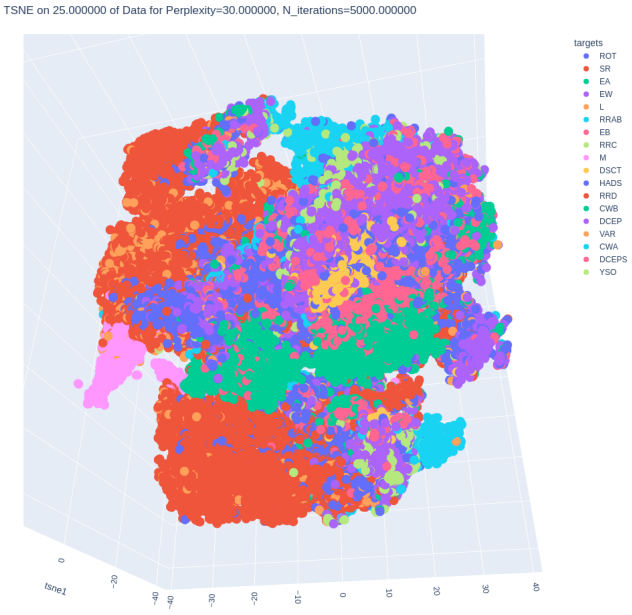
Applying K-Means, OPTICS, and DBSCAN in figure 15 also reveals a bit of a mess. K-Means seems like a homogeneous mix of clusters, which is a bit strange since as established in the theoretical description of the methods, we expect spherically-symmetric clusters. OPTICS shows over 3000 clusters also very mixed without any clear correspondence to the actual clusters. The results for DBSCAN are even more confusing, which reveal an almost complete assignment of every point to one cluster with one outlier. Trying values of $\min_{\text{pts}}$ from 6 to 50 reveal similar results. This indicates that the data are too close together spatially in the t-SNE representations. We also tried DBSCAN on the normalized (full 36-dimensional) Monster Matrix (MM) data, which revealed the same classification of all points in a single cluster with the exception of a few points classified as outliers. All of this illustrates that using Euclidean distance as a measure of spatial distance in both the full and compressed dimension case may not be optimal for our situation; we may, for example, consider the angle between vectors.

Something as well we noticed was that when generating lightcurves, many of the objects had magnitude errors $\sigma_i = 99.99$ mag, which is obviously an error. Therefore when we compute our variability indices, those that use σ_i in their definition are no longer liable to be valid.

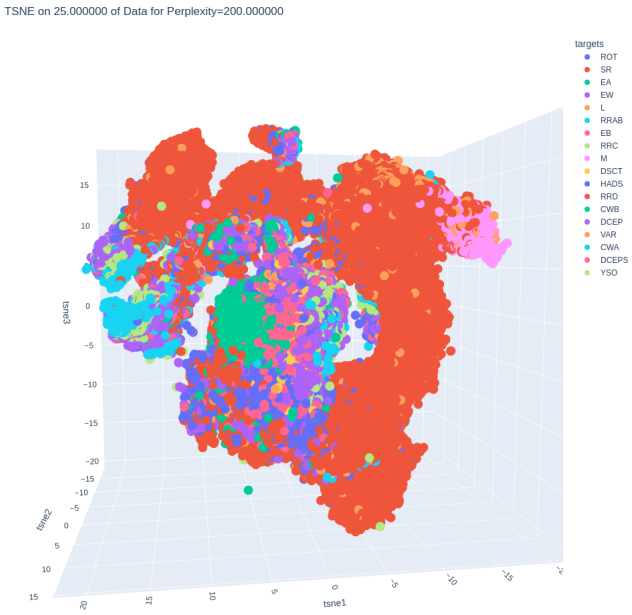
Moreover, when we plot sample lightcurves both in figures 18 and 19, which are three random objects that have MM period and ASAS-SN within ± 0.5 days and outside of ± 10 days respectively, we notice that the folded lightcurves using the MM period appear in a few cases good approximations, but overall do not capture the true shape of the ASAS-SN-folded curves. We investigated these suspicions by plotting the MM periods versus the ASAS-SN periods, and the results in figure 20a seem initially dismal. If our calculations reasonably matched those of ASAS-SN, then we would see a line with slope 1. Nevertheless, if take a random sample of $10,000$ points and compute their point density (within a neighborhood of ± 0.5 days around each point) as in figure 20, this reveals that once we rescale the axes and remove some noise ($\approx 8\%$ of data), we get a much clearer linear trend in 20c. However, you may notice vertical and horizonal bars in this figure. The vertical bar indicates a given short to null period from ASAS-SN but a nonzero period calculated by our data pipeline; the horizontal indicates the opposite. The density of objects reveals



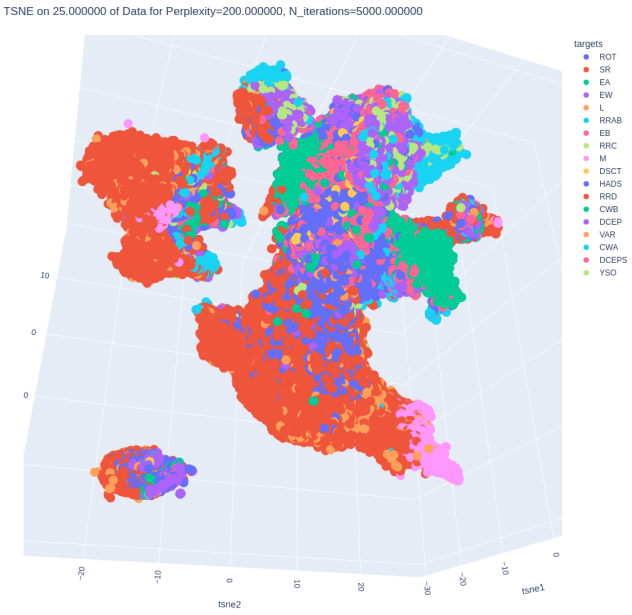
(a) t-SNE for perplexity 30 and $N_{\text{iterations}} = 1000$.



(b) t-SNE for perplexity 30 and $N_{\text{iterations}} = 5000$.



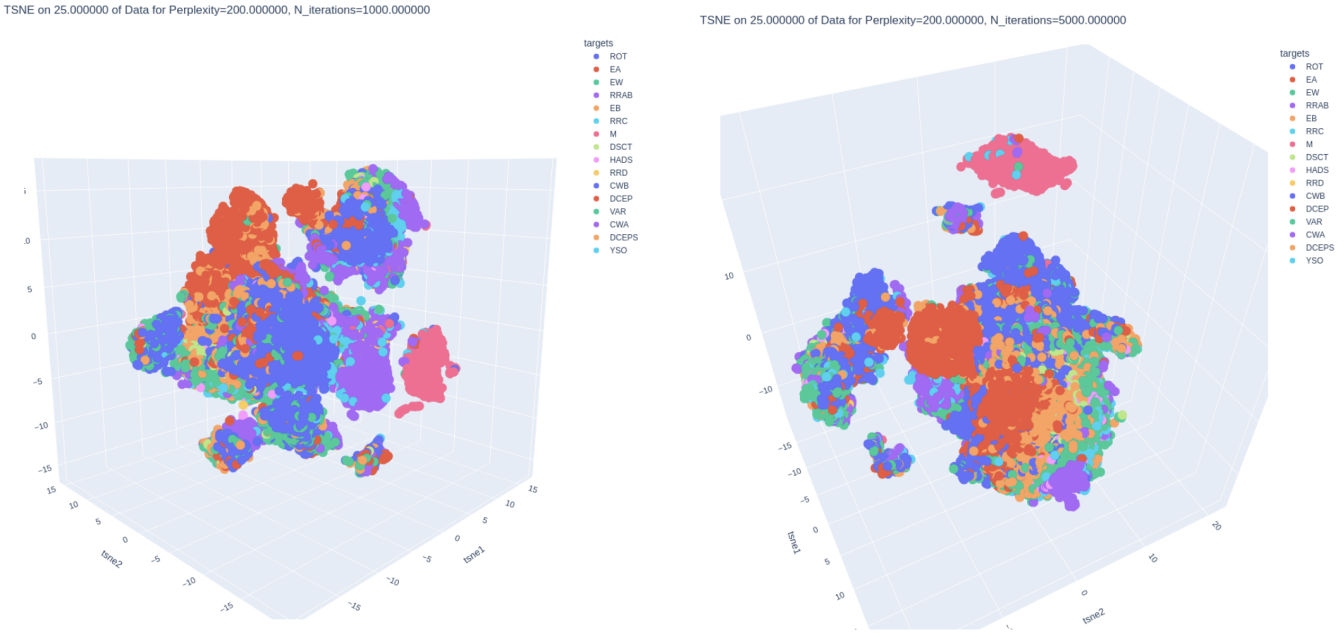
(c) t-SNE for perplexity 200 and $N_{\text{iterations}} = 1000$



(d) t-SNE for perplexity 200 and $N_{\text{iterations}} = 5000$

Figure 13: Plots showing the tSNE representations of all objects in the ASAS-SN catalog. Note the polarization of red/orange (aperiodic) with the menage of other colors (periodic) in the perplexity = 200 plots.

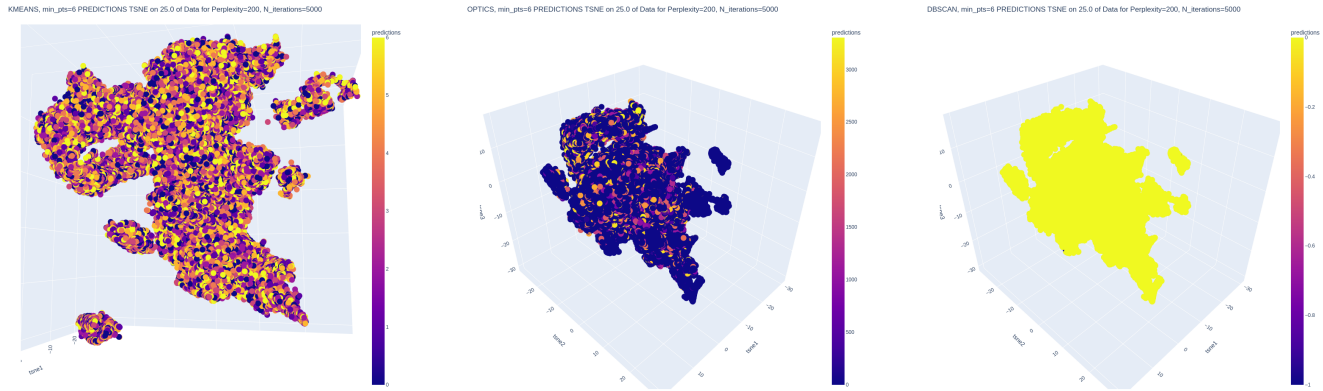
that there are far more of these vertical points than horizontal, which indicates our pipeline is incorrectly assigning periods to non-periodic objects. Using these random $10,000$ points as a representative of the system, we needed to remove $\approx 50\%$ of the data to achieve an R^2 of $> 0.95\%$—so perhaps around 50% of our periods were calculated incorrectly. Something we did not do but can implement is before assigning a period, we can check the returned False



(a) t-SNE for perplexity 200 and $N_{\text{iterations}} = 1000$.

(b) t-SNE for perplexity 200 and $N_{\text{iterations}} = 5000$.

Figure 14: t-SNE for only periodic type variables. Note the persistence of the pink "M island", which remains for all other combinations of hyperparameters we tried (perplexities of 75, 500, 100 for $N_{\text{iterations}} = 1000, 5000$ permutations for each perplexity).



(a) Results for K-Means.

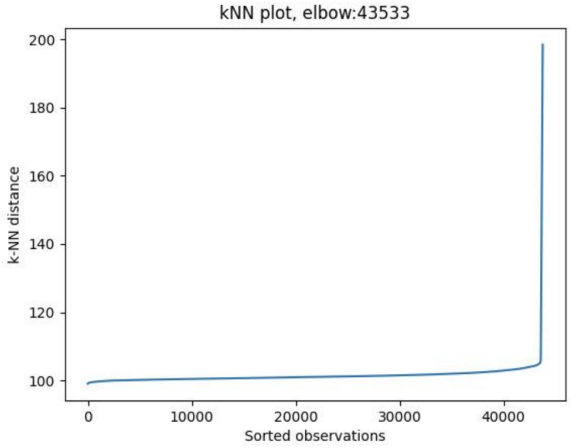
(b) Results for OPTICS.

(c) Results for DBSCAN.

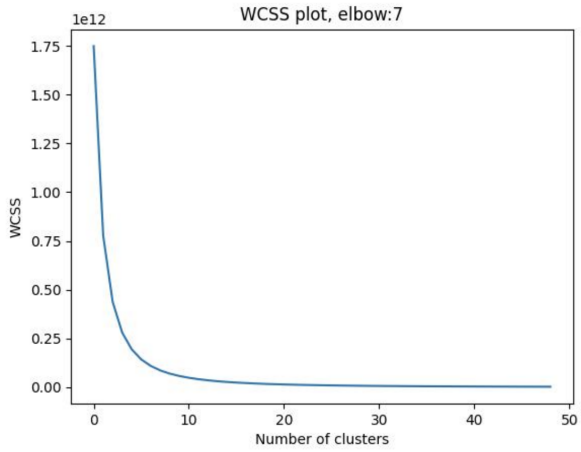
Figure 15: Results for unsupervised clustering using tSNE data with perplexity 200 and ϵ determined via a kNN plot.

Alarm Probability (FAP) generated from the Lomb-Scargle Periodogram and only assign the calculated period if the FAP is below a certain threshold value.

It is interesting to note that as we recall from the supervised results, adding the period values gave us a boost in accuracy. This supports our analysis that at least half of the periods are probably reasonably correct, otherwise, we would have expected overall a decrease in model performance. However, clearing up this issue with the MM period may help us with the unsupervised clustering problems. We will also need to continue investigating combinations of perplexities, $N_{\text{iterations}}$, and min_{pts} to improve the quality of data we feed into the unsupervised algorithms.

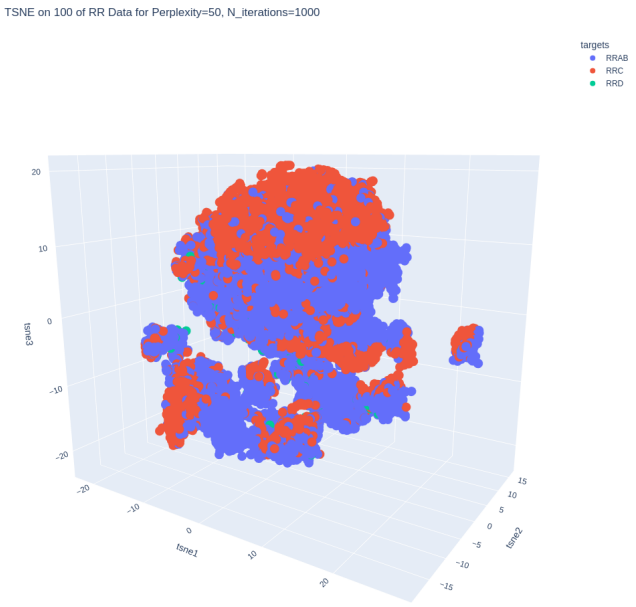


(a) Plot of k Nearest Neighbors given $min_{\text{pts}} = 6$ to determine ϵ for use in DBSCAN.

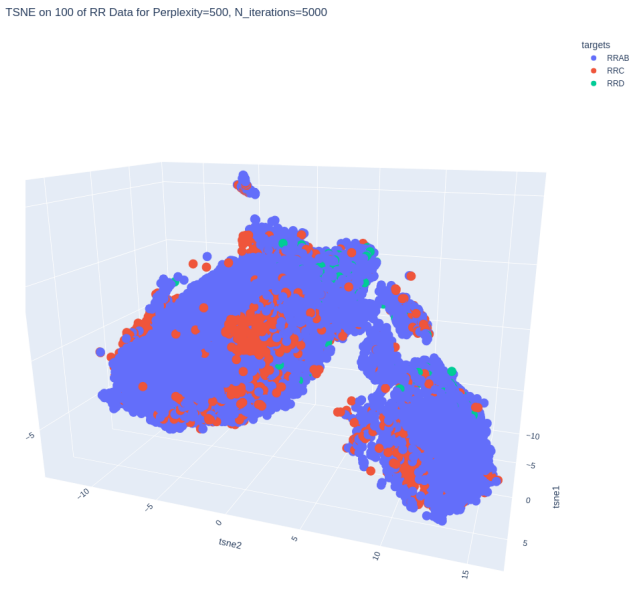


(b) Plot of the Within-Cluster Sum of Squares as a function of the number of clusters to determine optimal cluster number

Figure 16: Plots for determining the hyperparameters used in figure 15.



(a) t-SNE for perplexity 200 and $N_{\text{iterations}} = 1000$.

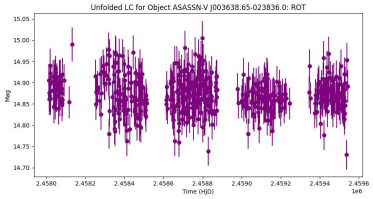


(b) t-SNE for perplexity 200 and $N_{\text{iterations}} = 5000$.

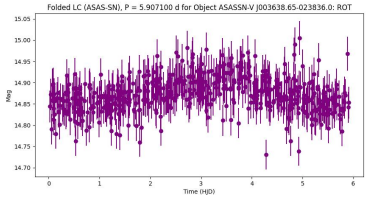
Figure 17: t-SNE for only RR-Lyrae type variables. Note that for $N_{\text{iterations}} = 1000$ there is somewhat of a distinction between RRC and RRAB, there is a lot of mixing and for $N_{\text{iterations}} = 5000$ it is even harder to distinguish.

The plots in figure 17 reveal, as with the other cases, a mixing of classes—however, the RRC and RRAB are more distinct than RRD. For $N_{\text{iterations}} = 1000$ this distinction is clearer, whereas with $N_{\text{iterations}} = 5000$ everything becomes more mixed. From the discussion above, we need to resolve issues with the period calculation and using t-SNE in general, so we will need to return to this question of identifying sub-clusters within larger clusters.

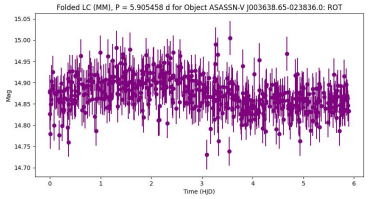
5. CONCLUSION



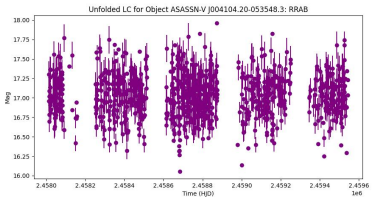
(a) Unfolded LC.



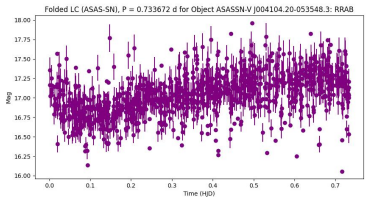
(b) Folded LC using ASAS-SN period.



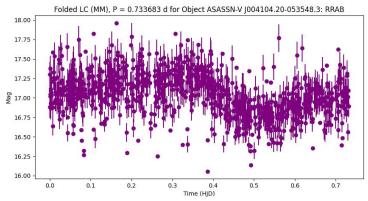
(c) Folded LC using MM period.



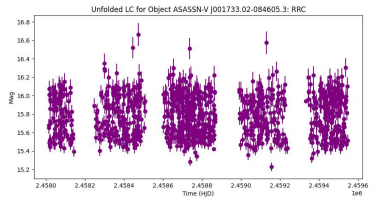
(d) Unfolded LC.



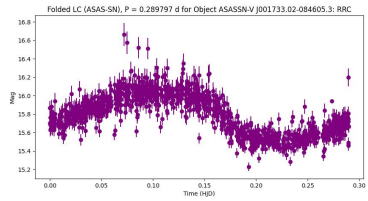
(e) Folded LC using ASAS-SN period.



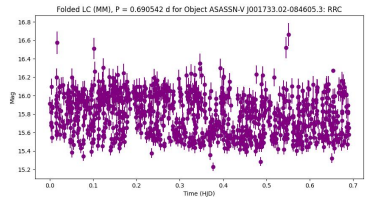
(f) Folded LC using MM period.



(g) Unfolded LC.



(h) Folded LC using ASAS-SN period.



(i) Folded LC using MM period.

Figure 18: Plots of lightcurves for |ASAS-SN period − MM period | ≤ 0.5 days. MM period is the period generated in the Monster Matrix from the mmgen2 datapipeline.

In summary, we were able to build a datapipeline for generating a Monster Matrix of 36 variability indices for use in supervised and unsupervised clustering networks. Our supervised model was able to achieve an accuracy of $> 85\%$ on new ASAS-SN data, which illustrates that our indices are a reasonably good metric for discriminating variable objects. As mentioned in table 3, we can consider more integer multiplies of LS_{Per} to probe lightcurve features at different harmonics. We can also resolve this inconsistency in period to hopefully further increase accuracy. For the unsupervised models, we need to further experiment with t-SNE in order to get a better sense of our data and how to project it to lower dimensions as well as better tune our unsupervised algorithms. Then we can revisit the question of discriminating sub-clusters in the ASAS-SN dataset and then apply this to the Marsh et al. (2017) data.

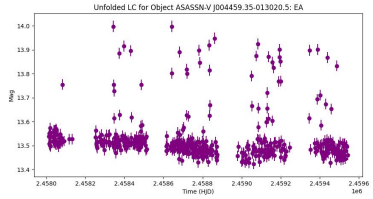
6. ACKNOWLEDGMENTS

We would like to thank Professor Choi for supporting this project this semester and offering us invaluable feedback along the way, as well as Professor Quetin for initializing us into the study of variable stars over Summer 2022. Also, I would like to thank my lab partner Graham Hirsch—we have both learned so much together, and I will be ever grateful for getting to know him and to explore neural networks together. #justearth

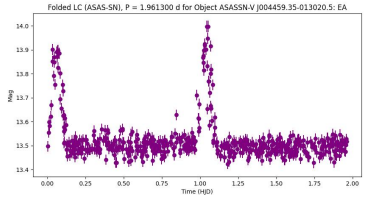
7. APPENDIX

Table 1. Definitions of Group 1 of the variability indices, adapting Pashchenko et al. (2018), Sokolovsky et al. (2017), Lopes et al. (2015).

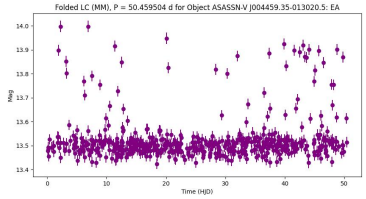
Group 1		
Name	Equation	Description
Median		Standard statistical measure of the magnitude at the 50% point in the data.



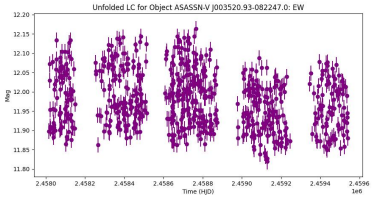
(a) Unfolded LC.



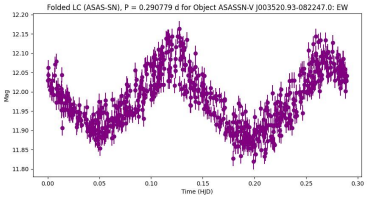
(b) Folded LC using ASAS-SN period.



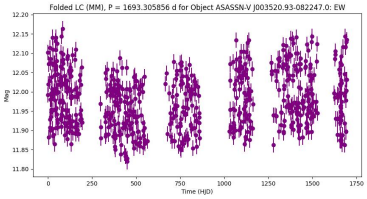
(c) Folded LC using MM period.



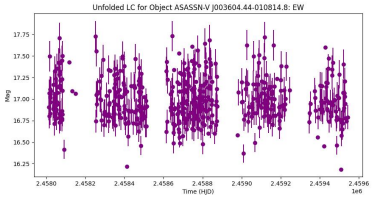
(d) Unfolded LC.



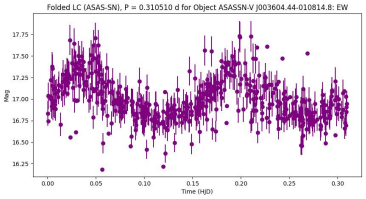
(e) Folded LC using ASAS-SN period.



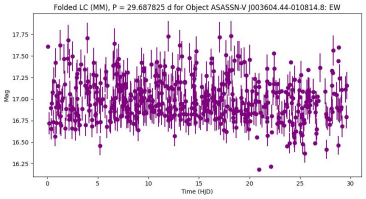
(f) Folded LC using MM period.



(g) Unfolded LC.



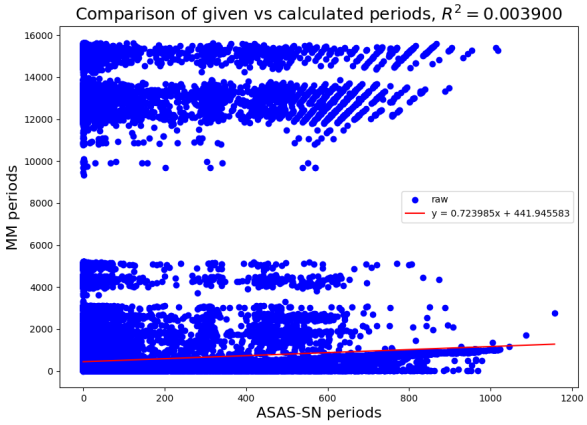
(h) Folded LC using ASAS-SN period.



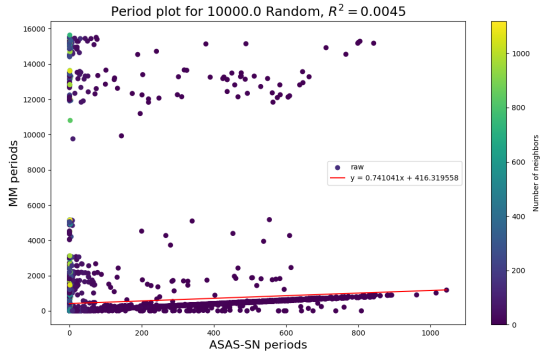
(i) Folded LC using MM period.

Figure 19: Plots of lightcurves for |ASAS-SN period − MM period | > 10 days. MM period is the period generated in the Monster Matrix from the mmgen2 datapipeline.

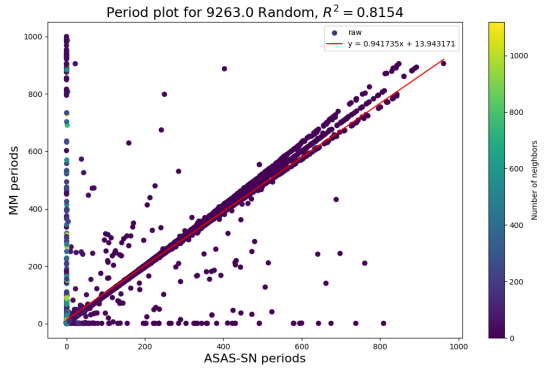
Continuation of Table 1		
Name	Equation	Description
Median Absolute Deviation	$\text{MAD} = \text{median}(\lvert m_i - \text{median}(m_i)\rvert)$	Insensitive to outliers, but equally insensitive to real variations that occur only infrequently.
Interquartile Range	$\text{IQR} = Q3 - Q1$	Robust measure of inner 50% of measurement values
Weighted mean	$\bar{m} = \sum_{i=1}^{N} \frac{m_i}{\sigma_i^2} / \sum_{i=1}^{N} \frac{1}{\sigma_i^2}$	More robust than standard mean because it assigns weighting based on magnitude error (σ_i).
Chi Squared Reduced	$\chi_r^2 = \sum_{i=1}^{N} \frac{(m_i - \bar{m})^2}{\sigma_i^2} / (N-1)$	Null hypothesis is no change in brightness. σ_i represents the magnitude error for each observation. Works well only if σ_i are accurate.
Standard deviation	$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (m_i - m_\mu)}$	Standard statistical measurement of the square root of variance; assumes equal weight to each observation. Also note m_μ is the unweighted mean.



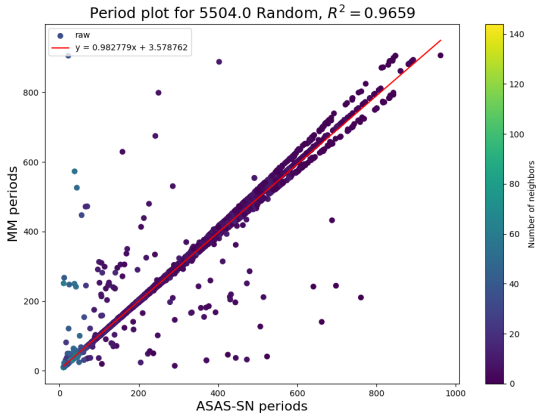
(a) Plot comparing all given and calculated periods.



(b) Random sample of 10,000 objects with density shown in color.



(c) Restriction for random sample with periods > 0 days on both x and y and a max cap of a 1000 day period.



(d) Restriction for random sample with periods > 10 days on x and y and a max cap of a 1000 day period.

Figure 20: Plots showing how our calculated periods (MM) compare with the given (ASAS-SN). Note that for the plots showing density of points, we have defined the nearest neighbor distance as ± 0.5 days. Note that the linear trend (a match between MM per and ASAS-SN period) is hidden in 20a, but the linear regression line does capture this result, albeit with a very poor R^2 value. Performing a random sample in 20b reflects a similar overall picture as 20a. As we impose restrictions on the period, a more clear linear trend emerges. Therefore even though it is initially hard to see in 20a, we verified that it does exist and hence partially validates our period calculations.

Continuation of Table 1		
Name	Equation	Description
Clipped standard deviation		Same formula as for standard deviation, but dropping the top and bottom 5% brightest and dimmest points. According to Pashchenko et al. (2018) σ_{clipped} is superior to σ as a variability index; also more sensitive than MAD and IQR to rare flashes/drops.

Continuation of Table 1

Name	Equation	Description
Weighted standard deviation	$\sigma_w = \sqrt{\frac{\sum_{i=1}^{N} w_i}{\left(\sum_{i=1}^{N} w_i\right)^2 - \sum_{i=1}^{N}(w_i)^2} \sum_{i=1}^{N} w_i \left(m - \overline{m}\right)^2}$	Note $w_i = 1/\sigma_i^2$ and σ_i is the magnitude error corresponding to m_i Using σ_i to reflect the relative accuracy of observations assuming Gaussian distribution. Note that standard deviation is in general sensitive to outliers, so it may be advised to pre-filter data before calculating values.
Robust median statistic	$\mathrm{RoMS} = (N-1)^{-1} \sum_{i=1}^{N} \frac{\lvert m_i - \mathrm{median}(m_i)\rvert}{\sigma_i}$	Expected value is 1 for non-variable object since majority of observations should be $\leq 1\sigma$ of median.
Normalized excess variance	$\sigma_{\mathrm{NXS}} = \frac{1}{N\bar{m}} \sum_{i=1}^{N} ((m_i - \bar{m})^2 - \sigma_i^2)$	Characterizes variability amplitude in presence of non-constant observation errors.; related to Power Spectral Density (PSD) slope.
Peak-to-peak variability	$\nu = \frac{(m_i - \sigma_i)_{\max} - (m_i + \sigma_i)_{\min}}{(m_i - \sigma_i)_{\max} + (m_i + \sigma_i)_{\min}}$	Sensitive variability indicator if no outliers are present. Monte Carlo simulation recommended to estimate expected ν for non-variable object given different number of observations.
Lag-1 Auto correlation	$l_1 = \frac{\sum_{i=1}^{N-1}(m_i - \overline{m})(m_{i+1} - \overline{m})}{\sum_{i=1}^{N}(m_i - \overline{m})^2}$	If we assume m_i are independent measurements, l_1 will follow as asymptotic normal distribution with expected value $\langle l_s \rangle = -\frac{1}{N}$ and $\sigma^2 \approx \frac{1}{N}$. Note that if the light curve is unevenly sampled, this method looses efficiency.
Welch-Stetson variability	$I = \sqrt{\frac{1}{n(n-1)}} \sum_{i=1}^{n} \left(\frac{b_i - \bar{b}}{\sigma_{b_i}}\right)\left(\frac{v_i - \overline{v}}{\sigma_{v_i}}\right)$	Captures degree of correlation between n quasi-simultaneous pairs of magnitude samples for two samples b, v. Note that we can generalize I to a lightcurve with only one filter by dividing it into odd and even measurements of n points each, but choosing $\bar{b} = \overline{v}$ as the mean of all $N = 2n$ observations.
Clipped I		Same formula as I, but before we form pairs of measurements, we check the difference in the date of observations against our threshold $T_{\max}$, which we set to 10 days in our analysis but can be specified when defining the Variable2 object.

Continuation of Table 1

Name	Equation	Description
Flux-independent I	$I_{fi} = \frac{1}{2}\left(\frac{1}{n_2}\sum_{i=1}^{n}\sum_{j=1}^{m-1}\sum_{k=j+1}^{m}\Lambda_{ijk}\right)$	Note m is the number of wavebands. Also, $\delta u_{ij} = \frac{u_{ij}-\overline{v_i}}{\sigma_{v_{ij}}}$ for each filter, and $n_2 = \frac{nm!}{2!(m-2)!}$ where n is the number of epochs. Moreover, $\Lambda_{ijk} = \begin{cases} +1 & \text{if } \delta u_{ij} > 0,\ \delta u_{ijk} > 0 \\ +1 & \text{if } \delta u_{ij} < 0,\ \delta u_{ijk} < 0 \\ -1 & \text{otherwise} \end{cases}$. Again, if we do not have multi-filter data we can split the data into even/odd pairs. The point of this index is to eliminate anomalous outliers that I does not account for.
Stetson's J	$J = \frac{\sum_{k=1}^{n} w_k \operatorname{sgn}(P_k)\sqrt{\lvert P_k\rvert}}{\sum_{k=1}^{n} w_k}$	Note w_k is defined above, sgn is the sign function. Moreover, $P_k = \begin{cases} \left(\sqrt{\frac{n_v}{n_v-1}}\frac{v_i-\overline{v}}{\sigma_{v_i}}\right)\left(\sqrt{\frac{b_v}{b_v-1}}\frac{b_i-\overline{b}}{\sigma_{b_i}}\right) & \text{P} \\ \frac{n_v}{n_v-1}\left(\frac{v_i-\overline{v}}{\sigma_{v_i}}\right)^2 - 1 & \text{SO} \end{cases}$ where P is a pair of observations in distinct filters b, v and SO is for single observations. Measures degree of correlation between consecutive brightness measurements.
Clipped J (J_{clipped})		Same equation as above, but see description for clipped I.
Time-weighted J (J_{time})		Same equation as J, but we weigh the pairs based on the time difference between observations: $w_i = \exp\left(-\frac{t_{i+1}-t_i}{\Delta t}\right)$. The advantage over J_{clipped} is this eliminates the need to fix an arbitrary $T_{\max}$.
Stetson's K	$K = \frac{1/N\sum_{i=1}^{N}\left\lvert\sqrt{\frac{n_v}{n_v-1}}\frac{v_i-\overline{v}}{\sigma_{v_i}}\right\rvert}{\sqrt{1/N\sum_{i=1}^{N}\left(\sqrt{\frac{n_v}{n_v-1}}\frac{v_i-\overline{v}}{\sigma_{v_i}}\right)^2}}$	Filter-paired observations synthesized to measure kurtosis, the peakedness of a distribution. For a Gaussian magnitude distribution, we expect $K \longrightarrow_{N\to\infty} \sqrt{2/\pi}$.
Stetson's L	$L = \sqrt{\pi/2}JK\left(\sum w/w_{\text{all}}\right)$	This index combines J, K to reduce L for stars with a fewer number of points to maximize likelihood of detecting mell-measured continously variable stars.
Time-weighted L		Same equation as for L, but we use J_{time} in place of J.
Excursions	$\text{Ex} = \frac{2}{N_{\text{scan}}(N_{\text{scan}}-1)}\sum_{i=1}^{N_{\text{scan}}-1}\sum_{j>i}^{N_{\text{scan}}}\frac{\lvert\,\text{median}_i - \text{median}_j\,\rvert}{\sqrt{\sigma_i^2+\sigma_j^2}}$	Here N_{scan} is the number of scans, where the data are split into a number of scans based on the $T_{\max}$ parameter mentioned above.
Inverse Von Neumann ratio	$\frac{1}{\eta} = \frac{\sum_{i=1}^{N}(m_i-\overline{m})^2/(N-1)}{\sum_{i=1}^{N-1}(m_{i+1}-m_i)^2/(N-1)}$	η quantifies the smoothness of the magnitude observations. We use $\frac{1}{\eta}$ since larger values would imply a greater chance of being variable, which is easier for the computer.

Continuation of Table 1		
Name	Equation	Description
S_B statistic	$S_B = \frac{1}{NM}\sum_{i=1}^{M}\left(\frac{r_{i,1}}{\sigma_i,1} + \frac{r_{i,2}}{\sigma_i,2} + \cdots + \frac{r_{i,k_i}}{\sigma_i,k_i}\right)$	Here N is the total number of observations, M is the number of groups of consecutive same sign residuals (CSSR) from a constant brightness model $r_{i,j} = \lvert m_i - \overline{m}\rvert$, where j is the running number in each CSSR group containing k_i entries, and $\sigma_{i,j}$ are the magnitude errors.
End of Table 1		

Table 2. Definitions of Group 2 (includes Group 1) of the variability indices, adapting Jayasinghe et al. (2020), and Christy et al. (2022a).

Group 2		
Name	Equation	Description
Lafler-Kinmann String Length (LKSL)	$T(t) = \frac{\sum_{i=1}^{N}(m_{i+1} - m_i)^2}{\sum_{i=1}^{N}(m_i - \overline{m})^2} \times \frac{N-1}{2N}$	We order the m_i by date of observation. The advantage of using LKSL is that it allows us to investigate the temporal structure of the measurements within the context of phase.
LKSL for Period ($T(\phi\|P)$)		Same equation as above, but here we order the m_i by sorting to a folded lightcurve using $LS_{\rm Per}$, so allows us to get a sense for how the folded lightcurve in the phase domain.
LKSL for Twice Period ($T(\phi\|2P)$)		Same equation as for (P), but here we sort the magnitude values by $2 \times LS_{\rm Per} \Rightarrow T(\phi\|2P)$. This allows for further exploration of features that may be apparent over a multiple of the phase—although Christy et al. (2022b) and we only considered this first multiple, we could in principle expand to a larger scope.
Difference for t and P	$\delta(t,P) = \frac{T(\phi\|P) - T(t)}{T(t)}$	Allows us to represent the difference in the statistic for unfolded and folded once to capture hidden features.
Difference for t and $2P$	$\delta(t,2P) = \frac{T(\phi\|2P) - T(P)}{T(P)}$	Allows us to represent the difference in the statistic for folded once and folded twice to capture hidden features.
Ratio of magnitudes brighter/fainter than average	$A_{\rm HL} = \frac{m_b}{m_f}$	Note $m_b > \overline{m}$ and $m_f < \overline{m}$ for all m_i. Here $\overline{m}$ is the unweighted average of magnitudes.
Skew	$\tilde{\mu_3} = \frac{\sum_{i=1}^{N}(x_i - \overline{x})^3}{(N-1)\sigma^3}$	Measures asymmetry of variable distribution about the mean (see Srivastav (2019)).
Kurtosis	$\mathrm{Kurt} = N\frac{\sum_{i=1}^{N}(m_i - \overline{m})^4}{\sum_{i=1}^{N}(m_i - \overline{m^2})^2}$	Similar in idea but with a different form than Stetson's K. If Kurt = 3, then we call the distribution Mesokurtic, that is normal (noa (2020)).

Continuation of Table 2		
Name	Equation	Description
2Mass Color1	$J - K_s$	Difference in color, computed using the 2 Micron All Sky Survey database (Skrutskie et al. (2006)) for the RA and Dec of each object. Variable stars are skewed towards the near-infrared, hence $J - K_s > 1$, while for constant sources $J - K_s \approx 0.5$. Theoretically, this makes sense because more evolved stars are cooler and more likely to be variable (Jayasinghe et al. (2020)).
2Mass Color2	$H - K_s$	Different color index, also in the infrared.
End of Table 2		

Table 3. Definitions of Group 3 (includes Group 2 and 1) of the variability indices, adapting Jayasinghe et al. (2020), and Christy et al. (2022a), Clarke (2002).

Group 3	
Name	Description
Best Lomb-Scargle Period ($LS_{\rm Per}$)	Used Astropy package to calculate the $LS_{\rm Per}$ corresponding to the lowest false alarm probability.
Power for $LS_{\rm Per}$	Returned by Astropy when calculating $LS_{\rm Per}$; statistical measure of likelihood of significance test (like $\chi^2_{\rm red}$) assuming a null hypothesis of constant magnitude detecting an effect. Hence a larger power corresponds to greater likelihood of variability.
End of Table 3	

REFERENCES

2020, Kurtosis Formula | Explantion, Example with Excel Template. https://www.educba.com/kurtosis-formula/

Al-Masri, A. 2022, Backpropagation in a Neural Network: Explained | Built In. https://builtin.com/machine-learning/backpropagation-neural-network

Anwla, P. K. 2021, K-Means. https://towardsmachinelearning.org/k-means/

Christy, C. T., Jayasinghe, T., Stanek, K. Z., et al. 2022a, The ASAS-SN Catalog of Variable Stars X: Discovery of 116,000 New Variable Stars Using g-band Photometry, arXiv, doi: 10.48550/arXiv.2205.02239

—. 2022b, Publications of the Astronomical Society of the Pacific, 134, 024201, doi: 10.1088/1538-3873/ac44f0

Clarke, D. 2002, Astronomy & Astrophysics, 386, 763, doi: 10.1051/0004-6361:20020258

Dhingra, S. 2021, Simplified Mathematics behind Neural Networks. https://towardsdatascience.com/simplified-mathematics-behind-neural-networks-f2b7298f86a4

Farm, M. N. P. 2022a, Iris virginica Native Blueflag, Wholesale Native Wetland Plants. https://midatlanticnatives.com/product/iris-virginica-native-blueflag-wholesale-native-wetland-plants/

Farm, W. G. 2022b, Iris setosa var setosa | Wild Ginger Farm. https://wildgingerfarm.com/plant-list/plants-i/iris-setosa-var-setosa.html#previous-photo

Flora. 2022, Gallery - Iris versicolor (blueflag) - Flora of Newfoundland and Labrador. https://newfoundland-labradorflora.ca/gallery/index.cfm?ParentID=47&alpha=i

Goyal, C. 2021, Guide to Prevent Overfitting in Neural Networks. https://www.analyticsvidhya.com/blog/2021/06/complete-guide-to-prevent-overfitting-in-neural-networks-part-1/

IBM. 2022, What is Deep Learning? | IBM. https://www.ibm.com/cloud/learn/deep-learning

Jayasinghe, T., Stanek, K. Z., Kochanek, C. S., et al. 2020, Monthly Notices of the Royal Astronomical Society, 491, 13, doi: 10.1093/mnras/stz2711

Kaggle. 2017, Iris Dataset. https://www.kaggle.com/datasets/vikrishnan/iris-dataset

Kumar, A. 2020, Keras - Categorical Cross Entropy Loss Function. https://vitalflux.com/keras-categorical-cross-entropy-loss-function/

Lopes, C. E. F., Dékány, I., Catelan, M., et al. 2015, Astronomy & Astrophysics, 573, A100, doi: 10.1051/0004-6361/201423793

Marsh, F. M., Prince, T. A., Mahabal, A. A., et al. 2017, Monthly Notices of the Royal Astronomical Society, 465, 4678, doi: 10.1093/mnras/stw2110

Melit Devassy, B., George, S., & Nussbaum, P. 2020, Journal of Imaging, 6, 29, doi: 10.3390/jimaging6050029

Pashchenko, I. N., Sokolovsky, K. V., & Gavras, P. 2018, Monthly Notices of the Royal Astronomical Society, 475, 2326, doi: 10.1093/mnras/stx3222

Sinclair, C. 2019, Clustering Using OPTICS. https://towardsdatascience.com/clustering-using-optics-cac1d10ed7a7

SKLearn. 2022a, sklearn.manifold.TSNE. https://scikit-learn/stable/modules/generated/sklearn.manifold.TSNE.html

—. 2022b, sklearn.decomposition.PCA. https://scikit-learn/stable/modules/generated/sklearn.decomposition.PCA.html

Skrutskie, M. F., Cutri, R. M., Stiening, R., et al. 2006, The Astronomical Journal, 131, 1163, doi: 10.1086/498708

Sokolovsky, K. V., Gavras, P., Karampelas, A., et al. 2017, Monthly Notices of the Royal Astronomical Society, 464, 274, doi: 10.1093/mnras/stw2262

Srivastav, A. K. 2019, Skewness Formula. https://www.wallstreetmojo.com/skewness-formula/

Wattenberg, M., Viégas, F., & Johnson, I. 2016, Distill, 1, e2, doi: 10.23915/distill.00002

Wikipedia. 2022a, k-means clustering. https://en.wikipedia.org/w/index.php?title=K-means_clustering&oldid=1124910014

—. 2022b, OPTICS algorithm. https://en.wikipedia.org/w/index.php?title=OPTICS_algorithm&oldid=1116998648

—. 2022c, DBSCAN. https://en.wikipedia.org/w/index.php?title=DBSCAN&oldid=1119633618

Wood, T. 2019, Softmax Function. https://deepai.org/machine-learning-glossary-and-terms/softmax-layer

Yıldırım, S. 2020, DBSCAN Clustering — Explained. https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556