Astronomy 101
A Tutorial for Supervised Clustering Neural Networks

Graham Hirsch

December 15, 2022

1 Introduction

Over the summer of 2022, my research partner Oscar Scholin and I were tasked with finding tidal disruption events within a large data set. Given that it was our first experience with this field and with research, we tried several different methods. Through this process, we learned basic coding skills and understood that the best way to complete this task was to create and use a Neural Network. At the time, we treated this like a black box; we fed it data, and it outputted either a helpful or unhelpful response. It was an incredibly inefficient way to go about conducting research. For my Astronomy 101 final project, I wanted to continue doing this work while explaining what exactly a neural network is, how to implement it correctly, and why it's useful. This tutorial will accomplish this, introducing code and a smaller data set so you can try everything while following along.

1.1 What is a Neural Network

Since the first implementation of computers in higher education, they have become a staple part of conducting research. They remove much of the tediousness of working with large data sets, allowing the researcher to process more data. Neural networks are a form of artificial intelligence extends this, interpreting the data, looking for patterns, and trying to make connections. This can be a disaster if the computer has no sense of direction or what

it's looking for, so researchers use machine learning techniques to train the machine.

There are millions of ways NNs have been implemented into society already. They are designed to operate like a human brain, and because they never tire, a researcher can send their program off into the depths of a data set to extrapolate exciting patterns[4]. It avoids the usual tediousness of data science while allowing one to process as much data as possible.

1.2 Neural Network Structure

Several types of NNs exist; however, I will focus on a feed-forward supervised clustering NN. That's a lot of words that essentially mean that we train our neural network on a portion of our initial data set to learn what patterns to look out for. We then validate it on another piece of the overall data set, and finally, we validate its accuracy on data it has never seen before. It's helpful to think of a NN in three stages:

1. Input Layer
2. Hidden Layer(s)
3. Output Layer

The input layer is the data that is initially read by the neural network. As you'll find out, this is a collection of different statistical parameters that help characterize the variability of our objects:

[0.1, 0.3, 0.6, 0.8]

The hidden layers are a collection of neurons that have different weights and biases associated with them. They aim to try and predict what star classification these statistical parameters are companions with and pass that prediction onto the next neuron.
The output layer will then be the same length as the number of classifications we're trying to predict. It will be filled with 0s except for a single 1, which is the predicted class the neural network is making:

[0, 0, 1, 0]

The NN thinks that this collection of data is a class 3 object.

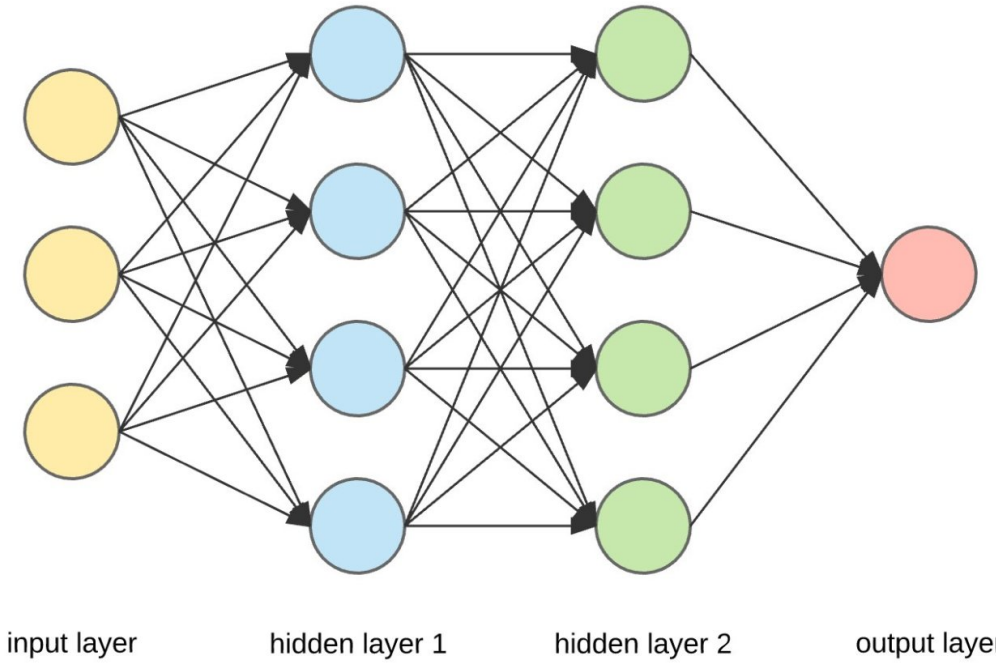


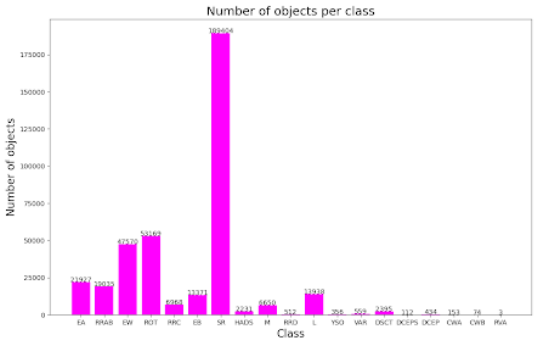
Figure 1: A typical image of a neural network. It's helpful to think of this in terms of a Plinko machine. Dropping the ball at the top is like feeding data into an input layer; it bounces through the hidden layers and ends up in a particular slot that defines what class this data point is. Thankfully though, neural networks aren't as random as Plinko is![3]

1.3 Introduction to the Data Set

The data set that we'll be working with is the ASAS-SN Catalog of Variable Stars[1].

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) Types of Variable Objects[1]



(b) Types of Variable Objects

Figure 2: This is a great data set to work with because of the sheer amount of data that they've included. There are about 370000 objects with 19 different classifications.[1]

We'll use a small subset of this data for the first run-through to minimize the computational power needed. It will be 500 objects from the top 5 most popular categories. This happens to be SR(semi-regular), ROT(spotted with rotational modulation), EW(W Ursae Majoris type binaries), EA(Detached Algol binaries), and L(irregular) type variables. This data set, titled "df_total_500", along with the original data set, can be found here[1] (click on the word here!).

1.4 Wait!!

Before we dive into anything more specific, it's essential to clarify what the point of this really is. If one reads through Christy et al.[1], it becomes clear that they used a neural network to classify their data! This paper is an excellent example of how NNs are helpful in picking out patterns in extensive data sets. Although this tutorial is on a smaller scale, it will have the same effect, helping us see the patterns in this data set and how to apply this method on a larger scale.

[1]In this tutorial, anything underlined is a link to this google drive folder, a python script, or another helpful website. Please click on them!

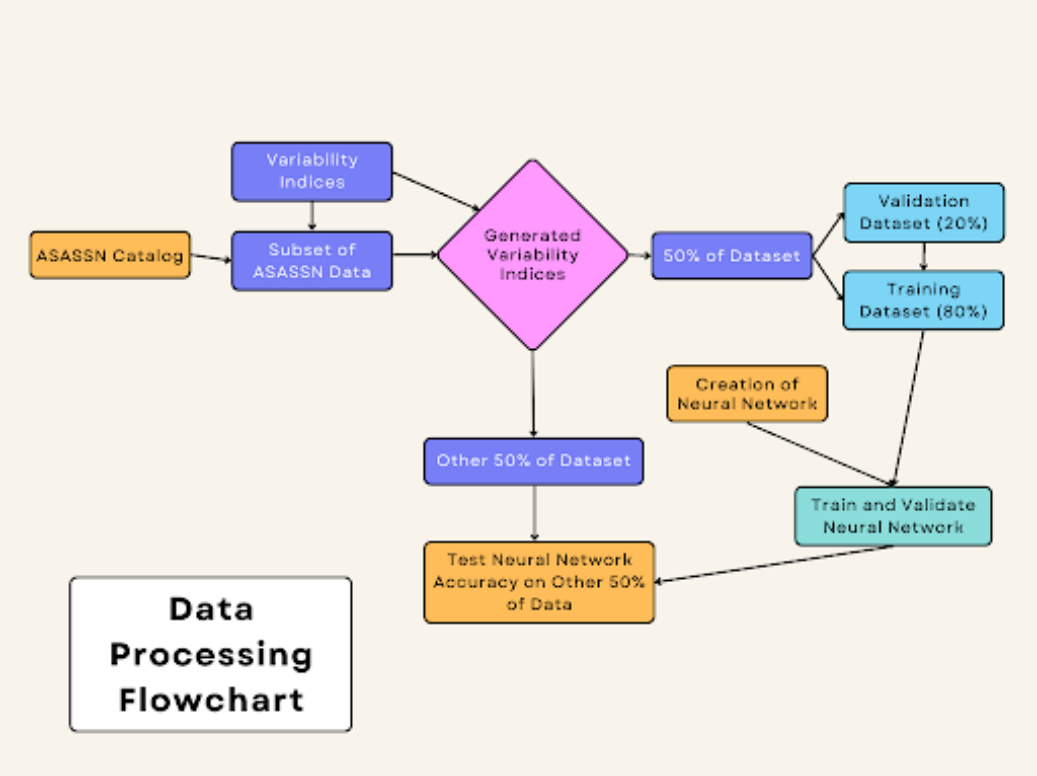


Figure 3: This is a model of the data reduction process. It may look daunting, but the actual process takes about 4-5 steps. In fact, most of the work goes into generating your variability indices.

2 Variability Indices

Hopefully, you've gone to the link that I've provided and downloaded the data set along with the .tar file that we'll be working with. The .tar file unzips a folder of .dat lightcurves for every object in our data set. It will be important for specific variability indices that use lightcurve data.

For clustering, we will use 13 statistics to further characterize our data. We will use a mixture of statistical and astronomical statistics to do this, helping our NN see the similarities and differences between sets of objects.

Implicitly, the more indices you have, the better your NN will describe your

data. However, because this is a tutorial, I want to clarify each index, how it works, and how to implement it. In this case, it's much better to start small and then build on the work that we've done.

2.1 Statistical Indices

I'm going to go through each statistical index that I use to characterize this data. You can check out and download the Python script here (document is called vargen_slimmed.py) to get a better idea of how they are implemented in code. It'd be helpful if you recreated each index on your own as that will maximize your understanding of how they function.

2.1.1 Weighted Mean

The weighted mean is very similar to the mean of a data set but is better at capturing variability. We calculate it as it's vital in calculating the reduced χ^2 test.

$$\bar{m} = \frac{\sum_{i=1}^{N} \frac{m_i}{\sigma_i^2}}{\sum_{i=1}^{N} \frac{1}{\sigma_i^2}} \tag{1}$$

2.1.2 Reduced χ^2 Test

A reduced χ^2 test is incredibly similar to a χ^2 test[2]. In this case, we'll use the weighted means of each object as our expected value and compare each data point to that and divide it by the number of degrees of freedom[3] in our data set.

$$\chi^2 = \sum_{i=1}^{N} \frac{(m_i - \bar{m})^2}{\sigma_i^2} \tag{2}$$

$$\nu = n - m \tag{3}$$

$$\chi_\nu^2 = \frac{\chi^2}{\nu} \tag{4}$$

[2]A χ^2 test detects how far away an observed quantity is from an expected value[2].

[3]Degrees of freedom (ν) are the number of independent variables that can be estimated. This ends up being the number of observations (n) minus the number of fitted parameters (m)

2.1.3 Weighted Standard Deviation, σ_w

The weighted standard deviation is how we measure the dispersion of values in our data set. In our case, a variable star should have more scatter than a non-variable one. Characterizing this scatter is one way to then find differences in object types. Again, we want to weight it to include any sort of estimated error.

$$\sigma_w = \sqrt{\frac{\sum_{i=1}^{N} w_i}{(\sum_{i=1}^{N} w_i)^2 - \sum_{i=1}^{N}(w_i)^2} \sum_{i=1}^{N} w_i(m_i - \bar{m})^2} \tag{5}$$

w_i is the weight which is equivalent to 1/(magnitude error)^2.

2.1.4 Interquartile Range

Like σ_w, IQR is another great way to measure the scatter of our data. It works especially well on distributions that are skewed. There's not an assigned equation for this, but it's essentially the third quartile of a specific statistic minus the first quartile of the same statistic[2].

$$IQR = Q_3 - Q_1 \tag{6}$$

2.1.5 Median Absolute Deviation

MAD, is yet another parameter that's great at measuring the scatter of observations. It first takes the median of a list and then subtracts that from a set of values. MAD is then the median of that list of values. In this case, we're working with lightcurves, so each m_i value is a different point with a set magnitude.

$$MAD = median(|m_i - median(m_i)|) \tag{7}$$

2.1.6 Normalized Excess Variance, σ^2_{NXS}

σ^2_{NXS} helps to characterize our variability amplitude when our measurement error is constantly changing[2]. This statistic is helpful because our data has been compiled from different sources.

$$\sigma^2_{NXS} = \frac{1}{N\bar{m}^2} \sum_{i=1}^{N} [(m_i - \bar{m})^2 - \sigma_i^2] \tag{8}$$

2.1.7 Peak to Peak Variability, v

This is an index that does exactly as it says, detects the variability in the peaks of our object.

$$v = \frac{(m_i - \sigma_i)_{max} - (m_i + \sigma_i)_{min}}{(m_i - \sigma_i)_{max} + ((m_i + \sigma_i)_{min}} \tag{9}$$

2.1.8 Skew and Kurtosis

Skew and Kurtosis are the measures of how asymmetrical our data is and how "tailed" our data is. They work well in tandem because if our data is right-skewed then it will have a tail shooting off to the right. These statistics will then characterize this tail in further detail.

2.2 Astronomical Indices

Hopefully, as you read through all the statistical indices, you can begin to see how they work to further characterize the variability of our data. This works incredibly well for lightcurves, where we have several points and are trying to consolidate them into a single number. Now, I'm going to go through each astronomical index that is used. These statistics are significant in characterizing stars while being reasonably simple to understand. Again, click here (called vargen_slimmed.py) if you're interested in seeing how this is coded.

2.2.1 Robust Median Statistic

RoMS is great at characterizing variability in an object. If an object is non-variable it's RoMS value is 1, therefore if σ is estimated correctly, then it's clear what objects are variable and which one's are not[2].

$$RoMS = (N-1)^{-1} \sum_{i=1}^{N} \frac{|m_i - median(m_i)|}{\sigma_i} \tag{10}$$

2.2.2 Clipped Standard Deviation

This gets rid of the brightest and dimmest sources from our lightcurve. Doing this helps to reduce any effect that outliers might have on our data.

2.2.3 von Neumann Ratio, η

This ratio is a bit difficult to explain. It's essentially the mean square successive difference to the distribution variance[2]. In other words, it essentially is an indicator of the smoothness of a lightcurve over a certain period of time. $\frac{1}{\eta}$ is yet another way to detect variability in an object and is what we use here.

$$\eta = \frac{\delta^2}{\sigma^2} = \frac{\sum_{i=1}^{N-1}(m_{i+1} - m_1)^2/(N-1)}{\sum_{i=1}^{N}(m_i - \bar{m})^2/(N-1)} \tag{11}$$

2.2.4 Lomb-Scargle Period and Power

Lomb-Scargle period is a way to calculate the period of an object over unevenly spaced observations. The Lomb-Scargle power is the frequency that best fits the given period, dividing this value by 1 gives you the Lomb-Scargle period[4].

2.2.5 Variability Detection Statistic, S_b

This statistic compares each point on a lightcurve to its residual, $|m_i - \bar{m}|$, doing this (along with some other math) is yet another way to characterize the variability of our objects.

$$S_b = (\frac{1}{NM})\sum_{i=1}^{M}(\frac{r_{i,1}}{\sigma_{i,1}} + \frac{r_{i,2}}{\sigma_{i,2}} + ... + \frac{r_{i,k_i}}{\sigma_{i,k_i}})^2 \tag{12}$$

2.3 Before Moving Forward

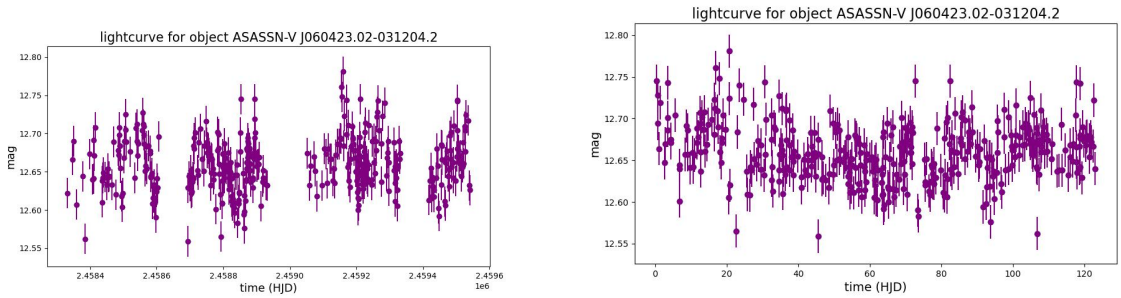
Hopefully, you've gone and downloaded or looked at the variability indices document (called vargen_slimmed.py) as you've followed through this section. This part of building a neural network is always the most difficult as this is where the data analysis is conducted. As my goal is to characterize my variable data set, I've chosen several indices that do a great job of doing this[5].

[4]To reiterate, this statistic gives one a solid estimation of the period when our observations are not consistent. If you have more questions or are confused about it's implementation in the Python script, check out the astropy documentation: `https://docs.astropy.org/en/stable/timeseries/lombscargle.html`

[5]If anything about this is confusing, I'd try to move forward and come back to this section. This makes the most sense after one has experience building NNs and can focus on this side of the data analysis

3 Generating Variability Indices

Now after all of this challenging work, it's time to generate our indices and create what Oscar and I refer to as a monster matrix[6]. This requires the use of another Python script (called mmgen-expl.py). Looking at the script, we begin by initializing all of our directories. Then, because we have data in both .dat form and in the CSV, we match the names in the CSV file with the names from the .dat folder and copy those .dat files to a new folder. We then fold the lightcurves based on their periods.



(a) Unfolded Lightcurve (b) Folded Lightcurve

Figure 4: This is an example of an unfolded vs. folded lightcurve. We want our indices to evaluate our objects as if the data had been taken consistently.

Finally, we initialize our new data frames, separating periodic and non-periodic variables, and generate our monster matrix. After this is developed, we normalize the data frame so that no term is seen as more or less dominant than another. The NN will weigh each index equally if every data point is between 0 and 1. This whole process should take around 10 minutes.

4 Neural Network Preparations

We're almost at the point in which we can create our neural network. If you look back at figure 3, you'll see that we have to split our monster matrix in half. This is so we can train and validate our data initially and then test

[6]This came from us playing around with statistical indices that had no business being in the same data frame as others. We would create these csv files so massive that we referred to them as monster matrices.

our accuracy on data that the NN has never seen before. This Python script does just that, along with creating the training and validation data itself. For this to work properly, we need training and validation data sets for both the inputs and outputs[7]. Therefore, while the inputs are the same as what's in the monster matrix, the outputs are one-hot encoded vectors, like the ones introduced in section 1.2 ([0,0,1,0]). Running this script will complete all the necessary prep work.

5 Building and Running the Neural Network

When building an NN, two primary hyperparameters need to be initialized: the number of layers and the number of nodes per layer. Experimentation is the best way to configure this; however, I've provided a great place to start here. This file is a Jupyter Notebook that allows you to take advantage of WandB. WandB is an application that is incredibly helpful when running NNs. For each parameter that is initialized, the NN chooses a range of numbers to see what weights give the highest accuracy. For example, let's say we have a neural network with one hidden layer choosing between 64 and 128 nodes. WandB will save whichever option got a higher accuracy score to your computer. It automates the daunting task of having to initialize each parameter while also saving that best model for you. For my model, I choose to use 5 dense layers[8], 1 dropout layer[9], another dense layer[10], an activation layer[11], and finally an optimizer[12] that tells the NN how to compile itself. Then, we train our neural network using the WandB agent and initialize the minimum and maximum sizes of each parameter. Again, I want to emphasize

[7]We split this into 80% for training and 20% for validation. You can split it into whatever percentages you like, this is just typical for NN training and validation data sets

[8]Dense layers is another way of saying hidden layers

[9]A dropout layer acts like pruning a tree, randomly dropping a small percentage of data.

[10]We call another dense layer here to reformat our data after we've just dropped some of it.

[11]The activation layer is how we measure the NNs accuracy. There are several different functions that you can use. If you want to find more information check out the Keras documentation here.

[12]Like activation functions there are several different optimizers that compile your data in different ways. Again, if you're interested, see the Keras documentation.

that I randomly choose the size of some of these layers [13]. Finally, we choose how many times we want the WandB agent to make a sweep (run the NN) and then we run the notebook!

6 The Output

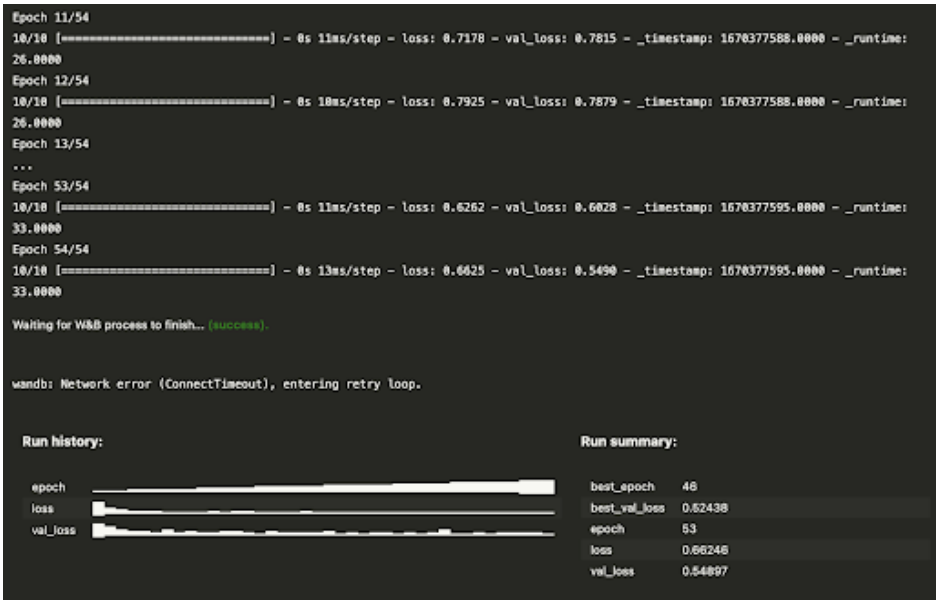


Figure 5: After waiting a few minutes, you should see something like this as an output. This tells us the number of epochs this sweep choose, the loss, which is what the NN is trying to reduce, and val_loss. The NN will adjust its weights to try and minimize these loss and val_loss values. Loss indicates how well the NN performed on data it's already seen, and val_loss suggests how well it performed on unseen data.

WandB will save the model with the lowest val_loss value as your best model. This can be found easily by logging into your WandB account, going to the project, and filtering for the sweep with the lowest val_loss value. Then click on that sweep and look at its overview to find the run path. Copy and paste

[13] It's recommended for computing power sake that when choosing numbers greater than 0, make sure they are powers of 2.

that, find it on your computer, and then enter that path into the model path for the final file from the Github. This file generates a confusion matrix to determine the overall accuracy of the NN.

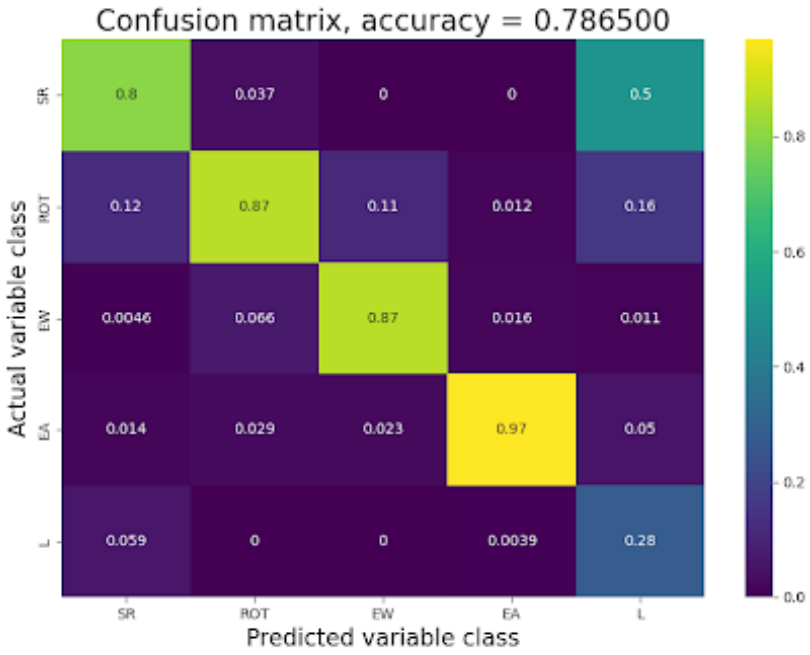


Figure 6: A confusion matrix compares the percentage of correctly predicted objects to their actual class. When I ran this neural network, I received the highest accuracy score at the top, about 79%. Look at the diagonals to see how many of a specific variable it's correctly matching. So, the NN does a great job of predicting EA, EW, ROT, and SR variables, while struggling to tell the difference between an L and SR variable. 50% of the L type variables it predicted were actually SR variables.

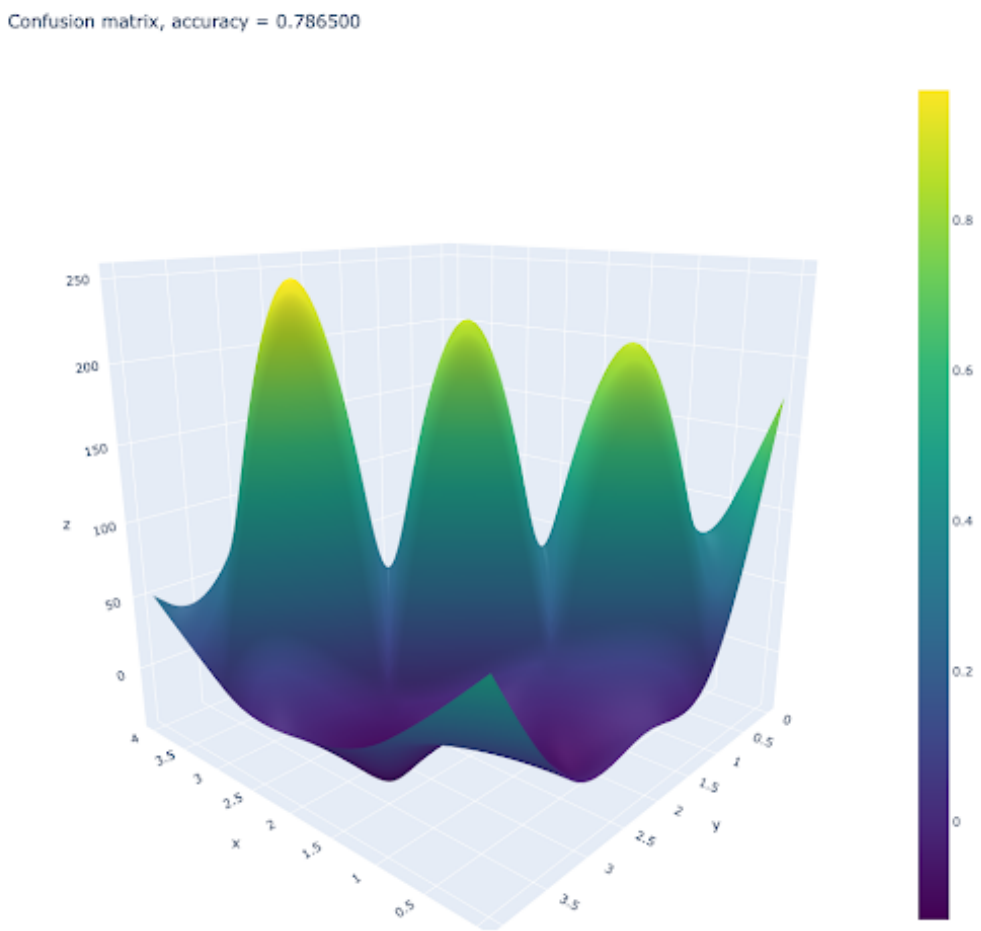


Figure 7: The file also creates a 3-dimensional confusion matrix to illustrate how much many more objects are in one classification compared to another. This is more useful when looking at the entire ASAS-SN data set.

7 Next Steps

Congratulations! You've finished this tutorial. Now, you should have a grasp of what a feed-forward supervised clustering neural network is doing, how to code it, and how to apply it to a data set. To increase accuracy, try and use

more variability indices or introduce more data. If you're interested in seeing the overall Github, click here.

References

[1] C.T. Christy et al. The asas-sn catalog of variable stars x: Discovery of 116,000 new variable stars using g-band photometry. http://arxiv.org/abs/2205.02239, May 2022.

[2] K.V. Sokolovsky et al. Comparative performance of selected variability detection techniques in photometric time series data. https://arxiv.org/pdf/1609.01716.pdf, 2017.

[3] Ayush Rastogi. Demystifying data-driven neural networks for multivariate production analysis. https://orgs.mines.edu/daa/blog/2019/08/05/neural-networks-mva/, 2019.

[4] Simplilearn. An ultimate tutorial to neural networks. https://www.simplilearn.com/tutorials/deep-learning-tutorial/neural-network, 2022.