Modelling RR-lyrae Probability Distribution Using Normalizing Flows

Abhinav Tyagi *

Abstract

Normalizing flows have been at the forefront of the research in machine learning in recent years. With normalizing flows comes the capability of modelling the exact probability distribution, which has found its applications in diverse fields, such as outlier detection and DNA modelling. The idea behind normalized flows is to transform a simple (usually normal) distribution to a multi-nodal distribution using a chain of bijection mappings (bijectors). The parameters for these bijectors can be trained using a neural network to achieve the desired distribution. In this paper, I build on the Lukas Rinder's implementation of neural spine flows (NSF) which is a class of normalizing flows in which the bijector used is a chain of polynomials defined on different regions (so called bins) of the domain. I implement this architecture to model the probability distribution of a 4 dimensional sample of background stars and RR Lyrae variables. RR Lyrae variables are the stars that show a periodic light curve and are instrumental in determining cosmic distances. I also build a RR Lyrae classifier which predicts the type of object, given the 4 color magnitudes, based on the simple probability ratios obtained from the implemented model. The classifier performs extremely well given the simple model, with 97% correct identifications.

1 Introduction

One of the most essential features of machine learning is to understand the probability distribution of the posterior. Maximum likelihood method maximizes this probability in order to return best-fit parameters for a given model. For most classification tasks, posteriors are assumed to have a Gaussian distribution, which, however, is not the case in most real life scenarios. Some solutions to this problem include implementation of adversarial training, i.e. likelihood-free inference such as Variational AutoEncoder (VAE) or Generative Adversarial Network (GAN). The exact evaluation and inference of the probability distribution is ,however, not possible with both of them. These models are also hard to train because of problems such as mode collapse and vanishing gradients posterior collapse [5]. The feature of normalizing flows that weighs over other similar methods is the capability of evaluating the exact probability distribution. This can be done using an invertible mapping (or bijection) $g : \mathbb{R}^d \rightarrow \mathbb{R}^d$. Let $\boldsymbol{x} \in \mathbb{R}^d$ be a random variable with a distribution $q(\boldsymbol{x})$. The probability distribution of the resulting random variable $\boldsymbol{y} = g(\boldsymbol{x})$ can be given as:

$$q(\boldsymbol{y}) = q(\boldsymbol{x}) \left| \det \frac{\partial g}{\partial x} \right|^{-1}$$

A normalized flow is obtained by applying a series of such bijections g_k. The expression $\det \frac{\partial g}{\partial x}$ is known as the Jacobian determinant and can be understood as the change of the volume under the probability distribution curve. With such bijections, a simple normal distribution can be transformed to a complicated multi-nodal one. The parameters of each transformation are tweaked with the help of a neural network to achieve a desired distribution.

1.1 RR Lyrae variable stars

In this project, I will be implementing normalized flows to determine the probability distributions of a sample of background stars and a special class of variable stars called RR Lyrae. Consistent and periodic change in the luminosity is a characteristic of variable stars. Various factors can contribute to periodic change of luminosity such as occultation by a star orbiting around the observed star in a binary system. But changes in luminosity of RR Lyrae's are associated with periodic expansion and contraction of external layers of these stars. The reason for this, although still debated, might be due to the phenomenon of pulsation, where the star constantly expands and contracts in a bid to maintain equilibrium due to counteracting radiation pressure gradients and gravitational forces [1].

Stars are often studied with photometric observations. This essentially involves measuring the brightness (relative magnitude, in astronomical jargon) of a star in different wavelength bands, denoted by u (ultraviolet - 365 nm), g (green - 464 nm), r (red - 658 nm), i (infrared - 806 nm), z (infrared - 900 nm). The difference of the relative magnitudes is called 'color' magnitude of a star because it denotes which wavelength band is its brightness more concentrated in. Its essentially a measure of the surface temperature of a star. In multi-color study of NGC 5272 M3 globular cluster [1], RR Lyrae variables were found to be concentrated in a narrow range of luminosity and color. They generally have a magnitude of roughly 0.6 and are thus about 50 times as bright as the sun [2]. Their short periods make their detection more probable which coupled with their near constant luminosity make RR Lyrae variables a feasible tool to measure distances in space.

2 Exploring the data

I use RR lyrae variables dataset provided in AstroML python library [6]. The dataset includes entries of 4 color magnitudes (u-g, g-r, r-z, z-i) for each star in a sample of about 93000 background stars and 500 RR lyrae variables. Figure 1 shows all possible combinations of the color magnitudes. While background stars are distributed over a wide range of color magnitudes, RR Lyrae variables are mostly found to be living in narrow regions inside the plot. This is consistent with the characteristics of RR Lyrae variables aforementioned.

*✉ s6abtyag@uni-bonn.de

3 Implementing the architecture

For this project, I essentially used Lukas Rinder's Implementation of normalizing flows [3]. He implemented different types of normalizing flows in Tensorflow 2.0. These include Masked Autoregressive Flows (MAF), radial flow, planar flow, Real-NVP and neural spine flow. In this paper, I have implemented the neural spine flow. This architecture, as implemented by Lukas Rinder is based on a paper by Durkan(2019) [4]. In this section, I will describe the most important aspects of this architecture taken from the paper.

3.1 Coupling transformations

As discussed earlier, each variable transformation using the bijection function g requires the computation of the determinant of the Jacobian determinant at that point. If the bijector function is chosen in such a way that the Jacobian matrix is lower traiangular, then the determinant is given by simply the product of its diagonal terms. Consider the transformation $\boldsymbol{y} = g_\theta(\boldsymbol{x})$. To achieve this, first $\boldsymbol{x}$ is split into $[\boldsymbol{x}_{1:d-1}, \boldsymbol{x}_{d:D}]$ and the first subset is fed into an arbitary neural network (NN) to determine the parameters $\theta_{d:D}$. Such dependence gives the aforementioned lower triangular jacobian. Found parameters are then used to determine $y_{d:D} = g_{\theta_{d:D}}(x_{d:D})$ index wise. Left over subset of variables is assigned in the following way: $y_{1:d-1} = (x_{1:d-1})$ for each index. Following the same steps, but instead using $\boldsymbol{y}$ as an input with the function g^{-1}, inverse can be calculated in a single pass as well. Such transformations are known as coupling transformations since they couple the input variables with each other to produce a certain output.

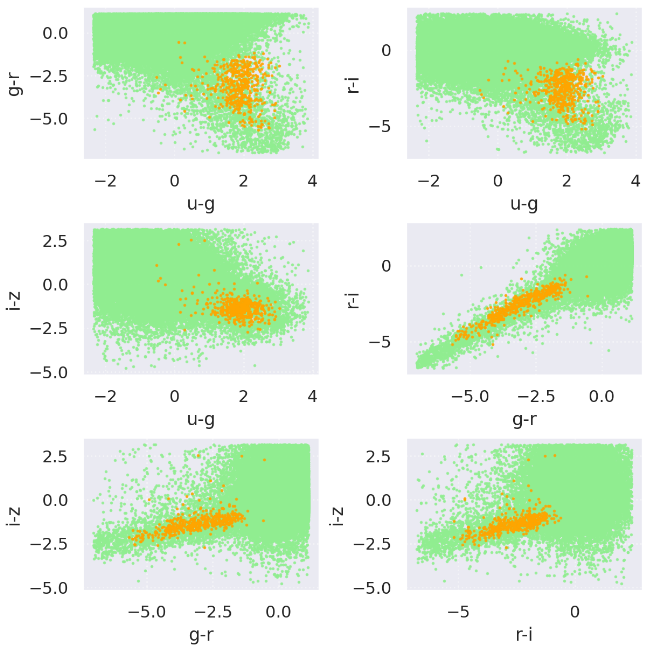

Figure 1: Green points are the background stars and orange points are the RRlyrae variables. All possible colors (represented as magnitude differences) possible for the four photometric magnitudes in the given data are plotted on the axes. The figure shows the narrow color ranges in which RR Lyrae variables appear.

3.2 Invertible transformations

In this particular model, the domain of the invertible function $g : [-B, B] \to [-B, B]$ is divided into $K + 1$ regions by K knots. A monotonic rational quadratic function is then defined between each two consecutive knots so as to keep the spline across the knots continuously differentiable. Monotonic rational quadratic splines are more flexible than simple quadratic splines and allow direct parametrization of the derivative and height at each knot. The polynomials are monotonic so as to keep them invertible.

Another important feature of the transformation is to ensure the permutation of the variables in $\boldsymbol{x}$ that are fed into neural network so as to teach the architecture about the interdependence of the variables.

To conclude the architecture, NN$(x_{1:d-1})$ spits a parameter vector $\theta_i = [\theta_i^w, \theta_i^h, \theta_i^d]$ for each i in $[d, D]$. The heights and widths of K bins are interpreted by parameters θ_i^h and θ_i^h respectively. The derivatives at the internal knots are given by θ_i^w. As g is an identity function outside the domain $[-B, B]$, the derivatives at boundary knots are set to 1 which ensures model stability as discussed by **Durkan**.

4 Implementation

I start by shuffling the data and splitting it into training, validation and test sample in the ratios of 80:10:10 respectively. Calculating the loss with the validation data helps the model to not get stuck on artifacts.

Normalization

For the maximum likelihood approach to work properly, it has often been observed that a normalized dataset is easier to train. The reasons for this are not very clear. It is however essential to normalize the data in case of Neural Spine Flow, because the splines are implemented in a region $[-B, B]$ wherein B is equal to the maximum or minimum of the data, whichever is larger in terms of magnitude. The architecture, therefore, assumes an inherent distribution around zero. To achieve this, I calculate the mean and the standard deviation for each colour index and implement the following:

$$\boldsymbol{x}_{\text{normalized}} = \frac{\boldsymbol{x}_{\text{initial}} - \mu}{\sigma}$$

Permutation

As discussed earlier, the architecture uses only a subset of variables as an input layer to the neural network that determines the parameter vector θ for the invertible function g. But for the model to learn the interdependence of different variables, I add a rotating variable layer after each bijector. The rotating layer essentially rotates the position of each variable by one place. Usually the permutation layer implemented in such algorithms, is using each half of the total number of variables consecutively as an input to the neural network. But since, the RR lyrae variables dataset has only four dimensions, such an approach does not implement the proper interdependence of variables. This resulted in strange unaccounted features in the produced probability distribution such as sharp cuts for certain values in earlier implementations.

Training

Finding an appropriate number of trainable parameters in a neural network remains a difficult task in machine learning. If the number of parameters is too small, they might not be able to depict the complexity of the model in hand but if they are too large, they might over-fit the data and therefore, perform terribly on the test data. In Durkan(2019), the author uses two coupling layers accompanied with 2 permutation layers and 128 knots (or bins) on the spline. This seems to have modelled a complex density distribution i.e. the facial features very efficiently. In addition to this, the number of neurons in the hidden layers of the neural network can also be adjusted. The RR lyrae variable dataset is a bit more complicated owing to its four dimensions. I tried various different values for these model variables. I observed that $4-8$ layers and 64-128 bins gave the most reasonable test losses, somewhere between $2.8-3.9$. The number of trainable variables becomes larger with increasing number of layers and therefore takes longer run time. Because of the extremely long time for a large number of layers, I could only perform systematic analysis for 2 coupling layers. The results can be seen in Appendix. In general, for a given number of coupling layers, as the number of bins increases, the number of neurons in the hidden layer should come down to give the least test loss. But I performed random tests with higher number of layers and found even lower losses. Hence for the classifier, the number of layers used was 8 with 64 bins (for stars I have implemented this only with 8 bins since the model with 64 bins was taking an extremely long time) and 2 hidden layers with 64 neurons each.

The flow is defined using tensorflow by connecting the bijector chain with a four dimensional standard normal. Training is done using stochastic gradient descent optimizer ADAM. The chosen batch size was 100 for RR Lyrae variables and 1000 for the stars owing to their respective sample sizes. Figure 2 shows the probability distribution of each color magnitude compared to the original data. The Kolmogorov-Smirnov test statistic is also shown in each subplot of the figure. It represents the maximum difference in the cumulative probability of the two distributions. As depicted visually, as well as through simple visual analysis, modelled probability distributions do a very good job of depicting the actual distribution. Scatter plots for visual comparison as plotted in 1 can be found in Appendix.

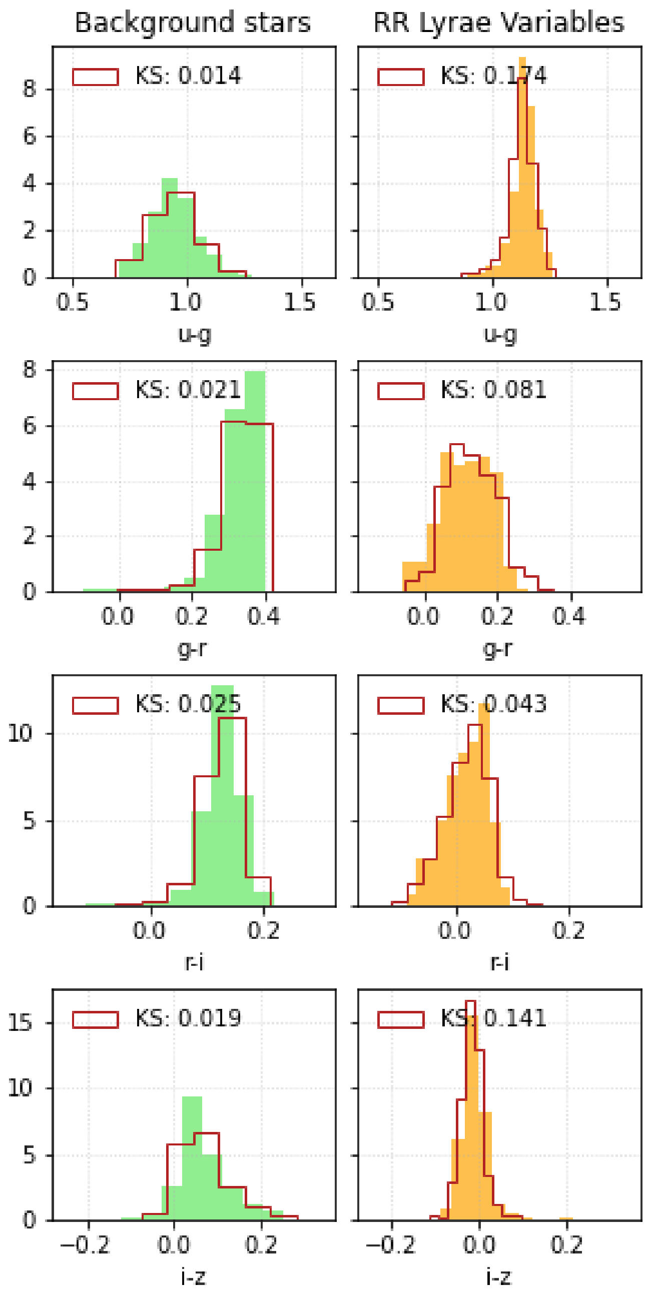

Figure 2: The figure shows the original data for the background stars (green) and RR lyrae variables (orange) in solid color histograms while the samples from the modelled flow are shown for respective stars in red color. For each set of distribution, Kolmogorov-Smirnov test statistic is also shown. The lower the value, the better is the resemblance between two distributions.

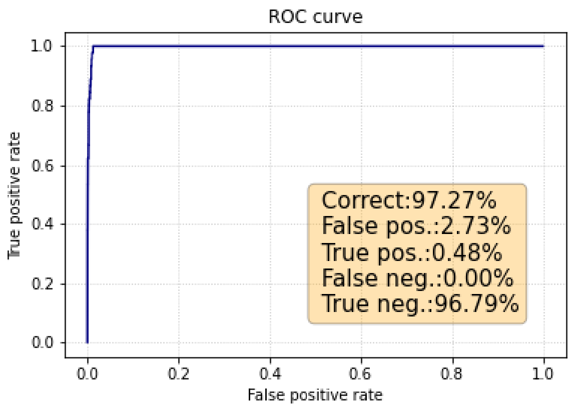

Figure 3: ROC curve. The figure gives the vital statistics from the implemented RR lyrae variables classifier.

5 RRlyrae classifier

Once the probability distributions of each color magnitude are obtained, they can be used to build a RR Lyrae variable classifier. This classification can be done by taking the ratio of the background stars probability distribution P_{stars} and the RR lyrae variables distribution P_{RR}. More precisely, an logistic activation function ϕ can be used on the ratio of the probabilities:

$$\phi(x) = \left(1 + \exp\left(-\log_{10}\frac{P_{\text{stars}}(x)}{P_{\text{RR}}(x)}\right)\right).$$

This activation function ensures sensitivity for a large range of ratios. Furthermore, I chose a threshold of 0.5 for the output of the activation function to classify a data entry as a star or an RR Lyrae variable. The results obtained from the classifier are shown in figure 3 along with the percentage of correct identifications of the stars either as background stars or RR lyrae variables. The classifier performs extremely well with around 97% correct identifications. The number of correctly identified RR lyrae variable is actually 99%.

6 Conclusion

In this paper, I built a Neural Spine Flow (NSF) based on Lukas Rinder's implementation of various normalizing flows. The model was built on a sample of background stars and RR lyrae variables. NSF modelled the probability distribution of both types of stars in the sample separately. I also built a RR Lyrae classifier which predicts the type of object, given the 4 color magnitudes, based on the simple probability ratios obtained from the implemented model. The classifier performs extremely well given the simple model, with 97% correct identifications.

7 References

1 Cacciari, C., T. Corwin, and B. Carney (2005). “A multicolor and Fourier study of RR Lyrae variables in the globular clus- ter NGC 5272 (M3)”. The Astronomical Journal 129(1), page 267.

2 Britannica. "Pulsating stars".https://www.britannica.com/science/star-astronomy/Pulsating-stars, Accessed on 28.08.2022

3 Lukas Rinder. "Normalizing flows".GitHub repository. https://github.com/LukasRinder/normalizing-flows.git

4 Durkan, C., Bekasov, A., Murray, I., Papamakarios, G. (2019). "Neural spline flows. Advances in neural information processing systems", 32.

5 Aryansh Omray. "Intoduction to normalizing flows". Towards Data Sceince. https://towardsdatascience.com/introduction-to-normalizing-flows-d002af262a4b Accessed on 28.08.2022

6 Vander Plas, J., Connolly, A. J., Ivezic, Z. (2014, January). "AstroML: Python-powered Machine Learning for Astronomy". In American Astronomical Society Meeting Abstracts 223 (Vol. 223, pp. 253-01).

8 Appendix

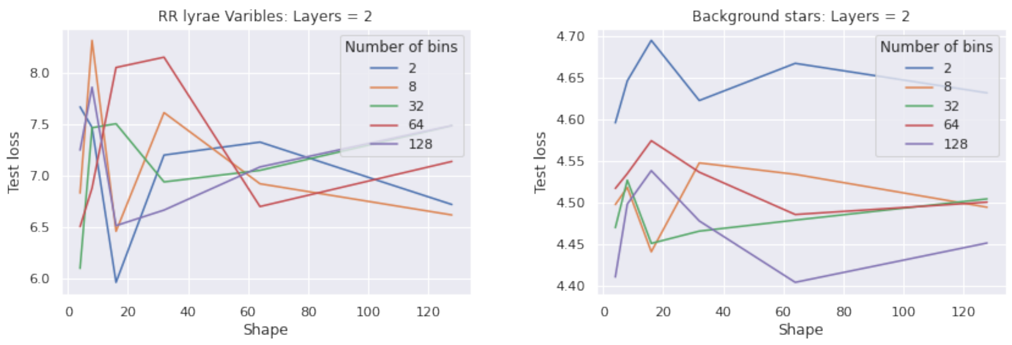

Figure 4: These figures show the systematic comparison of the test losses for different number of neurons in the hidden layers and different number of bins in the spline. As a general trend the minimum of test loss is achieved for stars for higher number of bins, whereas for RR lyrae variables it is achieved at a lower number of bins. This could be due to the much larger sample of background stars which provides sufficient data to the neural network to tune a large number of parameters.

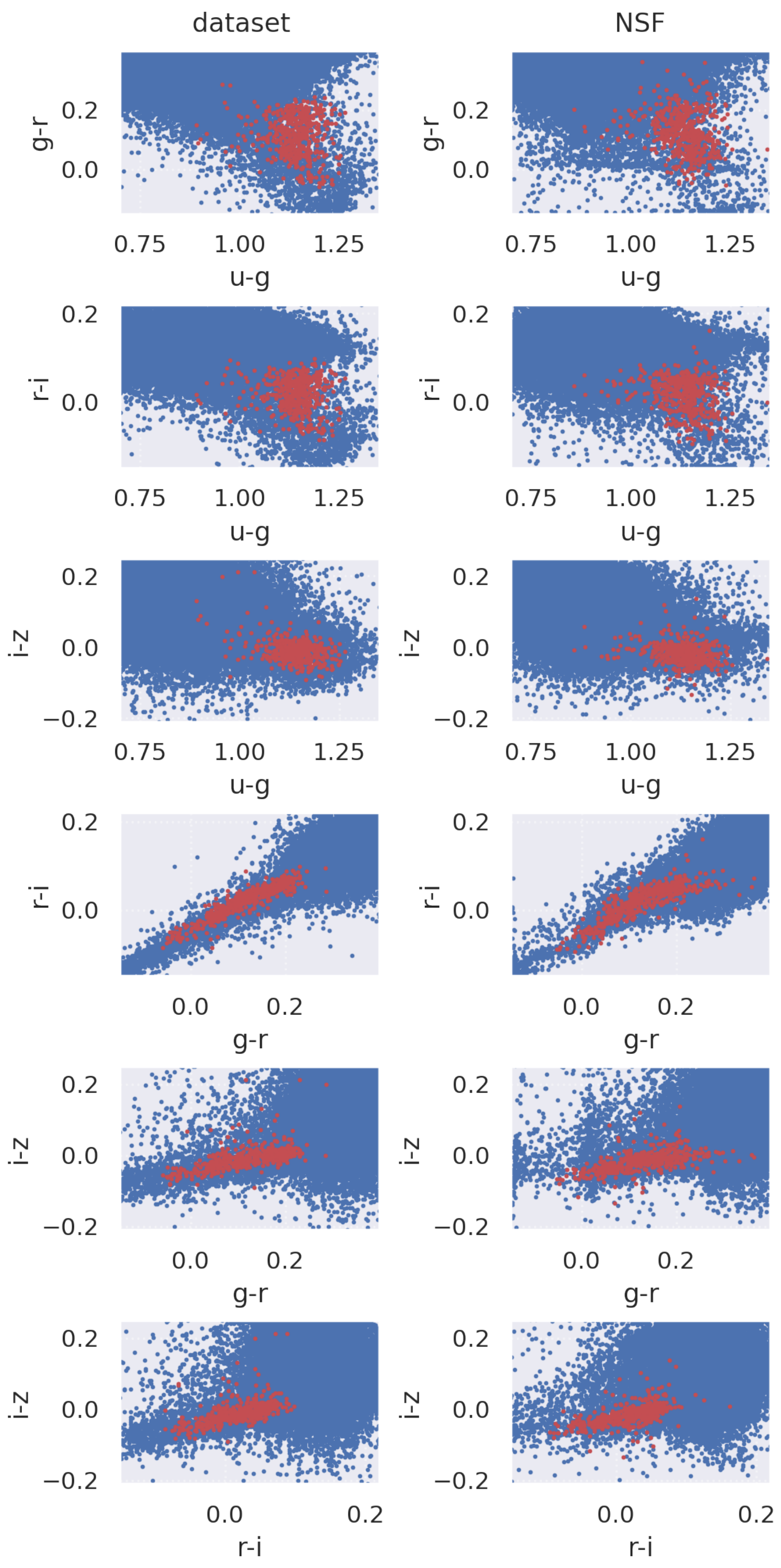

Figure 5: Visual comparison of the distribution of the samples from the dataset (left) and generated samples from the normalizing flow (right)