# Advanced Methods of Data Analysis

### Term paper 4: Normalizing flows

Lennart Böhm

**University of Bonn**

September 4, 2022

This term paper uses a type of neural networks, i.e. normalizing flows, in particular neural spline flows [1], to model the probability density function of a dataset of four colors of RR Lyrae stars and standard background stars from the SDSS [2]. The results are used to create a classifier for RR Lyrae. Based on its results, the performance of different normalizing flow architectures is compared, in particular the receiver operating statistic is used. The best classifiers reach true positive rates of 25 % and total correct classifications of 99.79 % for a dataset consisting of 0.56 % RR Lyrae.

## Contents

# 1. Motivation

Generative artificial neural networks (ANNs or just NNs) like the variational autoencoder (VAE) and generative adversarial network (GAN) show incredible results e.g. in the production of images (like fake faces [3]), but both have the disadvantages of being hard to train due to sensibility to the hyperparameters and having randomness in their architecture. This randomness prevents a particular input from being linked to a particular output and vice versa; mathematically speaking, they lack bijectivity, whereas normalizing flows (NFs) train variable transformations such that a base distribution like a multivariate gaussian is transformed to resemble the probability density function (pdf) of the input data. At each layer the flow is a normalized pdf and can be easily interpreted.

The aim of this work is to build a classifier based on pdfs from a labeled dataset. This approach would be considered as "supervised learning", but due to the modeling of the underlying pdf NFs are also very well suited for unsupervised learning, e.g. to find unknown classes in a dataset.

The code of the algorithms used for this work is largely based on a tutorial by Rinder [4] which uses the tensorflow python library.

# 2. Background

In the following, the general concept of NFs is explained first, followed by a brief explanation of how neural spline flows [1] (NFs). Finally, the used data set is described and the meaning of RR Lyrae stars is explained.

## 2.1. Normalizing flows

The concept of a normalizing flow can be roughly understood as a VAE, where the decoder is just the inverse of the encoder. We take a base distribution $\mathcal{Z}$ which is described by a pdf $p_Z$ (here a multivariate Gaussian) with the aim to learn a function $z = f(x) : \mathcal{X} \to \mathcal{Z}$ that transforms a variable $x \in \mathcal{X}$ from the data space to $z \in \mathcal{Z}$ in the latent space [5]. This function should be bijective to guarantee that its inverse $x = f^{-1}(z)$ can be formed and further be differentiable to provide a gradient for the minimization process during training [5].

Given a method to model the coordinate transformations (i.e. $f(x)$, see section 2.1.1) the function $f^{-1}(z)$ is trained and the density of the modeled distribution of $\mathcal{X}$ is (for the sake of computing power) evaluated as

$$\log \hat{p}_X(x) = \log p_Z(f(x)) + \log \left| \det \left( \frac{\partial f(x)}{\partial x^T} \right) \right|,$$

where $\frac{\partial f}{\partial x^T}$ is the Jacobian matrix of $f(x)$.

The computation time of the Jacobian strongly increases with the input dimension, therefore all of the examined architectures in this work rely on coupling layers [6]. Coupling layers split the $D$-dimensional input vector $\mathbf{x}$ in two vectors $\mathbf{x}_{1:d}$ of dimension $d$ with $d < D$ and $\mathbf{x}_{d+1:D}$ of dimension $D - d$. The vector $\mathbf{x}_{1:d}$ is used as the input of a NN to output the parameters $\theta$ of the transformation which then works on the elements of $\mathbf{x}_{d+1:D}$ like

$$f_{\theta_i}(x_i) = f\Big(x_i; \mathrm{NN}\left(\mathbf{x}_{1:d}\right)\Big). \tag{1}$$

Hence, in a coupling layer the input vector $\mathbf{x}$ is transformed to an output vector $\mathbf{y}$ by

$$\begin{aligned} \mathbf{y}_{1:d} &= \mathbf{x}_{1:d} \\ \mathbf{y}_{d+1:D} &= f_\theta(\mathbf{x}_{d+1:D}), \end{aligned}$$

from which it can be seen that the determinant of the Jacobian $\det \frac{\partial f_\theta}{\partial \mathbf{x}^T}$ simplifies to the product of its diagonal elements $\prod_{i=d+1}^{D} \frac{\partial f_{\theta_i}}{\partial x_i}$, which reduces computation time immensely. A detailed description is given in the papers ([6, 5, 7, 1]).

To ensure that all input dimensions can be taken into account for both, the parameters of the function and for its arguments, each coupling layer is followed by a permutation layer which permutes the output vector by a given arrangement. The permutation is also invertible which is important because it is part of the whole coordinate transformation.

By now we understand normalizing flows as coordinate transformations from a base distribution to the data distribution, where an exact likelihood exists at all times; unlike for VAEs, for example.

### 2.1.1. Neural Spline Flow

Based on the code implemented by Rinder [4], several architectures are build that differ in the type of coordinate transformation $f$ modeled. The best in terms of computational efficiency and results was the NSF.

A spline is a function that models a more complex curve by interpolating it with low order polynomials in successive bins. In this framework presented by Durkan et al. [1], the splines are based on $K$ monotonically-increasing, rational quadratic functions, ensuring bijectivity. The bins are defined by their $K + 1$ boundaries, called knots, at the coordinates $(x^{(k)}, y^{(k)})_{k=0}^{K}$ with the boundaries $(x^{(0)}, y^{(0)}) =$

$(-B, -B)$ and $(x^{(K)}, y^{(K)}) = (B, B)$, yielding $2K$ free parameters, i.e. the widths $\theta^{\mathrm{w}}$ and heights $\theta^{\mathrm{h}}$ of the bins. To ensure a stable log-likelihood training, jump discontinuities of the derivatives at the knots have to be prevented by fixing them to $\delta^{(k)}{}_{k=1}^{K-1}$, where $\delta^{(0)} = \delta^{(K)} = 1$ to ensure linearity outside the range $(-B, B)$ such that the NF is not restricted to it. This yields another $K - 1$ free parameters $\theta^{\mathrm{d}}$. The NSF paper [1] further elaborates how the monotonic, continuously-differentiable, rational-quadratic spline through the knots can be completely parameterized by the widths and heights of the bins and the derivatives at the knots.

The neural network $\mathrm{NN}(\mathbf{x}_{1:d})$ of input $\mathbf{x}_{1:d}$ which was mentioned in equation (1) here yields the $3K - 1$ parameters $\theta_i = [\theta_i^{\mathrm{w}}, \theta_i^{\mathrm{h}}, \theta_i^{\mathrm{d}}]$ for each dimension of the output vector $\mathbf{x}_{d+1:D}$. How the initially unconstrained parameters get constrained to the interval $[-B, B]$ and other details can be see in Durkan et al. [1].

The most important parameters to tune the NSF are the number of bins, the number of layers and the shape of the NN that yields the $\theta_i$, while the arrangement of the permutation will only play a role if the number of layers is less than the input dimension.

## 2.2. The dataset

To develop and test a classifier data from the Sloan Digital Sky Survey (SDSS) [2] is used. The SDSS measured the magnitude of stars (and other astronomical objects) in five frequency bands around the optical, i.e. a filter for the ultraviolet (u), green (g), red (r) and two in the infrared (i and z). The colors are defined by the subtraction of the magnitudes in two consecutive filters.

The data is provided by the datasets package from the astroML [8] Python library and contains the four colors (u-g, g-r, r-i, i-z) of 93 141 stars and an array of ones and zeroes marking those that are observed to be RR Lyrae stars (483, i.e. 0.5 %). That means we have a $93\,141 \times 4$ array of (binary) labeled data.

The 2D distribution of the data is shown in figure 1, where each scatter point corresponds to one star. Due to the large number of standard background stars I chose to additionally plot density contours for them, such that their distribution becomes better visible while preserving the comparability between them and RR Lyrae. This reveals that there are more background stars in the RR Lyrae region than RR Lyrae themselves, making it hard to detect them if they are not labeled.

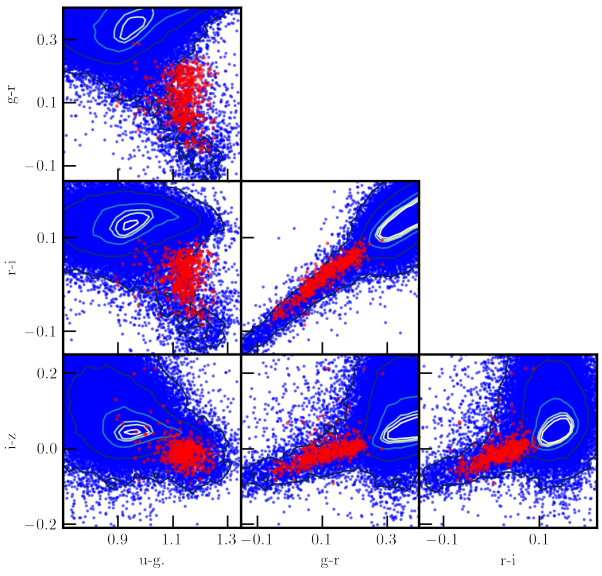


Figure 1: All possible combinations of colors of RR Lyrae (red) and standard background stars (blue) from the SDSS, provided by the `astroML` [8] python library. The contours show the density of the standard background stars which is not represented well by a scatter plot due to their large number

### 2.2.1. RR Lyrae stars

For some general understanding it should be noted that RR Lyrae stars are a special kind of variable (pulsating) star. Variable stars like Cepheids and RR Lyrae show a period-luminosity relationship which enables astronomers to determine their absolute magnitude after some calibration. Schneider [9] explains how absolute magnitude, along with apparent measured magnitude, is a practical tool for determining distances in astronomy. RR Lyrae have the shortest periods, which is due to the fact that they have low masses [9].

They are abundant in globular clusters, the bulge and the halo of the Milky Way. In fact, they were an important tool to describe the almost spherical shape of the halo (H. Shapley 1917, [10]) and locate its center around 8 kpc from the sun in the constellation of Sagittarius[1] [10].

[1] There, the central supermassive black hole could be imaged just recently by the EHT collaboration [11].

# 3. Architecture, training and results

With the papers to the given architectures I felt confident to work with the code by Rinder [4]. In order to stay within the scope of this paper, only the best performing architecture is described, but in the final result, all that were examined are compared. Some more details are to be found in appendix A.

## 3.1. Architecture

As suggested by the task sheet, the NSF performed best. Performance is understood as a mixture of computational effort and the classifier statistics. With the aim to build a good classifier, a good statistic to rate the architecture is the area under the receiver operating characteristic (ROC) curve, i.e. the plot of true positive rate over false positive rate. Figure 3 in appendix A shows how these statistics differ very little; the five best architectures classified $(99.75 \pm 0.04)\,\%$ of the data correctly and their areas under the ROC curve (AUC) differ by less than $0.1\,\%$. Although the cleanest classifier is given by a MAF, a NSF wins the best performance because while the other architectures had to train at least 90 min, the NSF needed less than 20 min. To achieve this performance, many parameters had to be tweaked.

Due to the low input dimension $D = 4$ there is no other useful option to split the vector in the coupling layer than half the input dimension, hence $d = 2$.

The value of $B$ for the interval $(-B, B)$ in which the spline gets defined is chosen to be the ceil of the maximum of the absolute of the data $\mathbf{x}$, i.e. $\lceil \max |\mathbf{x}| \rceil$. That way, if $\mathbf{x}$ is asymmetrical, lots of the data is not covering the range of the spline. I received better results if this was not the case, hence I normalized the data in a first "layer" by taking its mean $\mu$ and standard deviation $\sigma$ and applying

$$\mathbf{x}^1 = \frac{\mathbf{x}^0 - \mu}{\sigma}\,.$$

This is done for each of the four dimensions.

**Permutation** The code by Rinder [4] used to just flip the first half of the vector with its second half, which mixes the dimensions too little[2]. Therefore a rotating permutation was used, taking the last element of the vector and putting it in the first position, moving every other element one position further.

**Bin number** In the NSF paper [1], two coupling layers with 128 bins were sufficient to model even complex pdfs which shows that a large number of bins is more important than a large number of layers. The training time is also not increased that much by bins (the output vector of the NN in equation (1) just gets longer), hence an architecture with 64 bins is used, although it should be noted that there was no systematical check of all parameter configurations possible.

**Layer number** The depth of the network, i.e. the number of layers, can improve the result, but it has a considerable influence on the training time. In addition, the optimization via the gradient becomes more and more difficult with increasing depth. I therefore chose a small number; with only 4 layers, the pdf of the RR Lyrae could already be modeled enormously well.

**Shape of the NN** The number of hidden layers and knots in the NN from equation (1) can be changed, but the shape of two hidden layers with 64 knots each as used by Rinder showed the best performance, hence I did not change it.

## 3.2. Training

The previously described layers (neural spline followed by a permutation) are in opposite order put in a list and assembled into a chain of bijectors[3]. The flow is then defined as a transformed distribution object[4] which connects the bijector chain with a four-dimensional standard normal. It gets trained with the stochastic gradient descent optimizer Adam[5], where it is given batches of 100 data points for the RR Lyrae and 1000 data points for the standard background stars.

The training needs to be stopped at some point. I choose to validate every 10 epochs, i.e. every 10th time the optimizer saw the full dataset, if the loss is still decreasing, i.e. if it is improving. When the loss has been increasing for more than 15 epochs, the code stops and uses the parameters from the minimum loss.

To avoid the optimizer getting hung up on artifacts in the data, the training progress is not validated on the same dataset that is being trained on. Therefore, it should be briefly mentioned that before the training,

[2] The flipping permutation resulted in incorrect samples of the data, which could be circumvented by the rotating permutation (see appendix B).

[3] tensorflow_probability.bijectors.Chain()

[4] tensorflow_probability.distributions.TransformedDistribution()

[5] tensorflow.keras.Adam

the data was randomly[6] split to 80 % training data, 10 % validation data and 10 % test data. The training data, which test the classifier at the end, remain mixed from RR Lyrae and standard background stars, but to learn the different pdfs, training and validation data are separated into the two classes to train two different flows.

## 3.3. Classifier

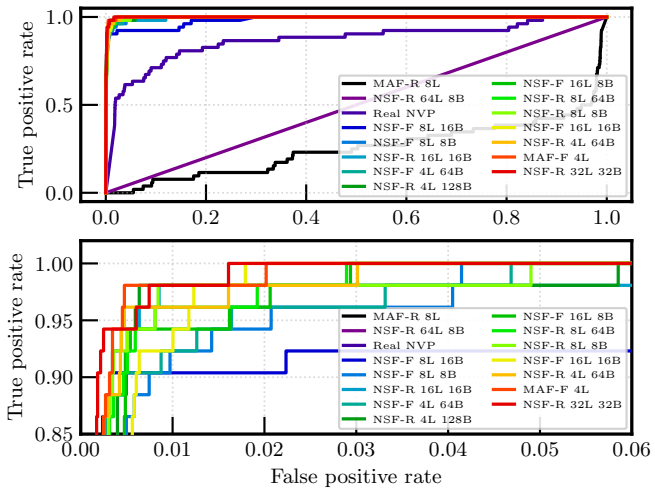


Figure 2: Receiver operatic characteristic curves of the tested architectures with a zoom on the region close to the perfect classifier in the bottom plot. The number with a L is the number of layers, i.e. 8L corresponds to 8 layers, and the number with a B (e.g. 8B) indicates the number of bins. The letters after the architecture type refer to the permutation pattern, i.e. a flipping (F) or a rotating (R). The architectures are ordered from worst to best

The flow-object offers a method to evaluate the probability of any given input vector $\mathbf{x}$. With the learned pdfs, both probabilities, i.e. their multiplied values at $\mathbf{x}$, for the standard background stars $p_0(\mathbf{x})$ and the RR Lyrae $p_1(\mathbf{x})$ can be compared. The classification is done by taking the logarithm of their ratio $\log_{10} \frac{p_1(\mathbf{x})}{p_0(\mathbf{x})}$, which maps the ratio to zero in case of equality, to $-1$ if $p_0(\mathbf{x}) = 10 \cdot p_1(\mathbf{x})$ and so on. This can than be interpreted easily by an activation function $\phi$ as was done in exercise sheet 5 of this class. To be sensitive even for ratios up to 10 I choose the logistic function[7]

$$\phi(\mathbf{x}) = \left(1 + \exp\left(-\log_{10} \frac{p_1(\mathbf{x})}{p_0(\mathbf{x})}\right)\right)^{-1},$$

for which then a threshold in the range $(0, 1)$ to classify a RR Lyrae star needs to be chosen. Before this will be done for the two cases of a generally good classifier and for a clean classifier, the overall strength of it can be shown by a ROC curve which is basically a plot of the true positive rate (tpr) over the false positive rate (fpr), or in other words sensitivity over one minus specificity. Both are calculated using the function `roc_curve` from the scikit-learn library and the result is shown in figure 2. The figure compares the best-performing architecture described here with other architectures examined, as elaborated in appendix A. The perfect classifier corresponds to a point in the top left corner of the ROC plot, i.e. a tpr of 1.0 with a 0.0 fpr.

With very little computational effort, the optimal threshold is determined by checking 10 000 equally distributed values between 0 and 1. Depending on the desired strength of the classifier, different thresholds are found, so at a threshold of 0.894 the classifier identifies the largest fraction of all data points correctly. From the test data block, 99.70 % are correctly assigned to a class; there are 0.56 % RR Lyrae in the test data block. For research, however, the RR Lyrae found would not yet be useful, because while 77 % of them are found (i.e. the tpr), up to 40 % of the as RR Lyrae classified stars are actually standard background stars, i.e. two in five RR Lyrae would be false. This sample might be too dirty for scientific purposes.

For a particularly clean classifier, the ratio of correctly to falsely as RR Lyrae classified stars should be maximized. To do this, 10 000 thresholds between 0 and 1 are tried as before. At a threshold of 0.9633 an optimal result is shown with a tpr of 11.54 % and an fpr of 0.00 %, thus roughly one in nine RR Lyrae is found and all found RR Lyrae are definitely to be understood as such. This is a great success, but figure 4 shows that the MAF finds even cleaner samples with a tpr of 25.00 % at zero fpr.

Thus, it could be shown that NFs are very good tools for modeling complex pdfs. In addition to the low computational effort, they could also convince by the simpler structure compared to other generative NNs.

[6] A fixed random seed for the sake of comparability between the architectures has been used.

[7] It was after the coding that I noticed the simplification due to choosing the natural logarithm to

$$\phi(\mathbf{x}) = \left(1 + \frac{p_0(\mathbf{x})}{p_1(\mathbf{x})}\right)^{-1}.$$

# References


[1] Conor Durkan et al. “Neural spline flows”. In: *Advances in neural information processing systems* 32 (2019).

[2] Branimir Sesar et al. “Light curve templates and galactic distribution of RR Lyrae stars from Sloan Digital Sky Survey Stripe 82”. In: *The Astrophysical Journal* 708.1 (2009), p. 717.

[3] Tero Karras, Samuli Laine, and Timo Aila. *A style-based generator architecture for generative adversarial networks*. `https://github.com/NVlabs/stylegan`. 2019.

[4] Lukas Rinder. *Normalizing Flows*. `https://github.com/LukasRinder/normalizing-flows`. 2020.

[5] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. “Density estimation using real nvp”. In: *arXiv preprint arXiv:1605.08803* (2016).

[6] Laurent Dinh, David Krueger, and Yoshua Bengio. “Nice: Non-linear independent components estimation”. In: *arXiv preprint arXiv:1410.8516* (2014).

[7] George Papamakarios, Theo Pavlakou, and Iain Murray. “Masked autoregressive flow for density estimation”. In: *Advances in neural information processing systems* 30 (2017).

[8] Jacob VanderPlas et al. “Introduction to astroML: Machine learning for astrophysics”. In: *2012 conference on intelligent data understanding*. IEEE. 2012, pp. 47–54.

[9] Peter Schneider. *Extragalactic astronomy and cosmology: an introduction*. Vol. 146. Springer, 2006.

[10] Albrecht Unsöld, Bodo Baschek, and Wolfgang J Duschl. *Der neue Kosmos: Einführung in die Astronomie und Astrophysik*. Springer, 2002.

[11] Kazunori Akiyama et al. “First Sagittarius A* Event Horizon Telescope Results. I. The Shadow of the Supermassive Black Hole in the Center of the Milky Way”. In: *The Astrophysical Journal Letters* 930.2 (2022), p. L12.

[12] Francisco Villaescusa-Navarro et al. “Cosmology with one galaxy?” In: *The Astrophysical Journal* 929.2 (2022), p. 132.

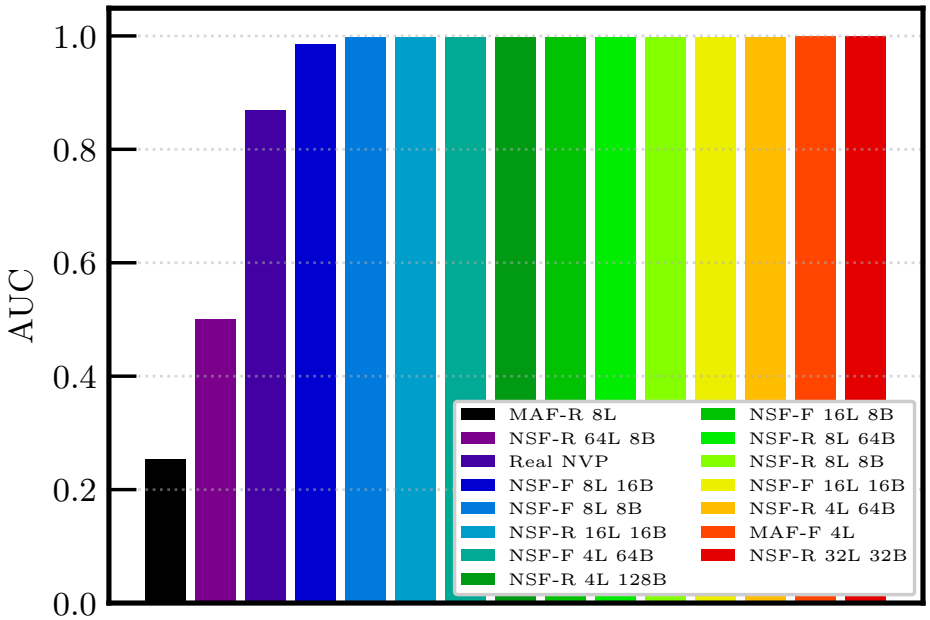


Figure 3: Area under the receiver operating characteristic curve taken as a measure for the performance of the classifiers. The figure nicely shows how many of the classifiers perform almost equally good

While working on the term paper some ideas beyond the scope of the exercise led to results which will be shown briefly in the following.

## A. Other architectures

Next to the NSF a Real NVP and MAF have been implemented, where the MAF performed better with the flipping permutation (and all other parameters like Rinder set them). But one change was important: I had to remove the batch normalization layers even though the MAF paper [7] states to receive better results with them, while doing one intial normalization to mean and standard deviation. For the Real NVP I did the same. Further the MAF used a lot of RAM with the same batch size like the NSF and calculated even longer (105 min) than the 32 layer and 32 bin NSF (90 min). The Real NVP [5] showed the worst performance if one ignores the 64 layer NSF which is just a random classifier (ROC curve is a straight line) and the 8 layer MAF with rotating permutation that is even worse than a random classifier (I suspect a bug in the code). When comparing the performance of the architectures the right statistic should be chosen with respect to what the NF will be used for. In this work it was used for a classifier, therefore I used the area under the ROC curve as as a measure, which is shown in figure 3. As the best five architectures differ in the AUC by less then 0.1 % it is hard to go into more detail. Therefore I specified the goal to a clean classifier with zero fpr. Figure 4 zooms on this region and the MAF with 4 layers shows its strength. Second and third cleanest classifier were achieved with the NSF with rotating permutation and 8 layers with 8 bins as well as 4 layers with 128 bins. More data on how the different architectures performed as a overall correct and as a clean classifier is given in table 1.

## B. Sampling

When it comes to sampling useful statistics might be the Kolmogorov-Smirnov (KS) test to show if the hypothesis that two samples are taken from the same pdf can be ruled out with 95 % confidence. The KS test relies on continuous distributions, where for discrete distributions the Epps-Singleton test statistic would work better. Sadly I could not get good results from any test statistic for the sample of standard background

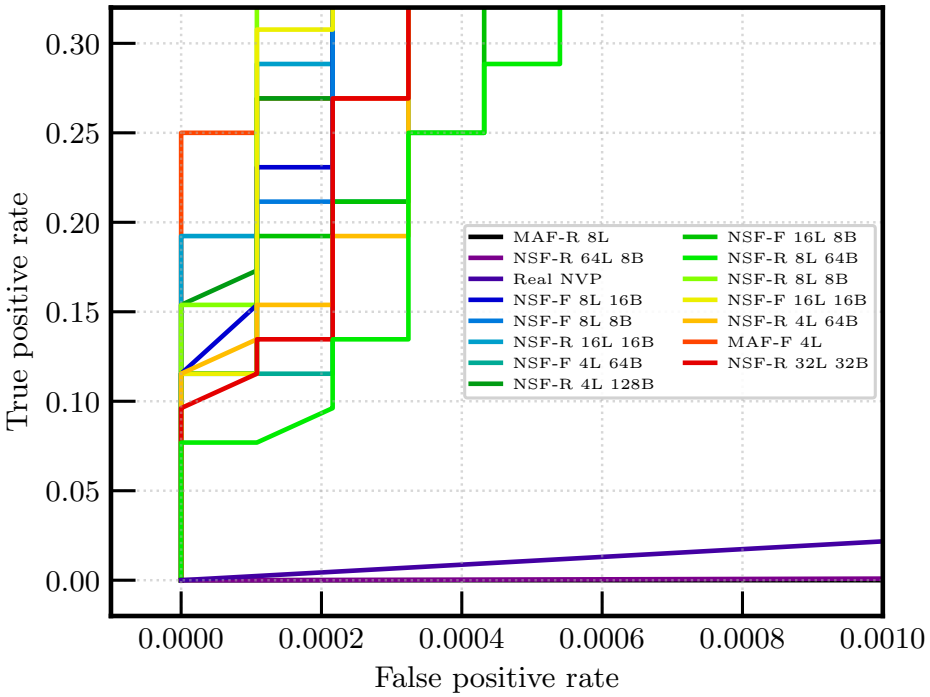


Figure 4: Zoom on the region of the ROC curves where the fpr is zero, corresponding to a perfectly clean classifier. The plots are ordered in a way that it becomes well visible where they leave the zero fpr value, with the best one the MAF

stars even when the histograms looked very promising, but for the RR Lyrae it was confirmed to be the correct pdf. Figure 5 and figure 6 show the histograms used to compare the samples, which are not able to reveal a major problem with the flipping permutation. Therefore the distribution of the samples are shown in two-dimensional plots like in exercise 1 in figure 7. There, the contour show the densest regions of the stars, where the long structure in the region of the RR Lyrae is only thinly populated. The NSF-F architecture is not able to correctly model this thin region, hence it is completely cut off in the sample. The NSF-R architecture does not have this weakness. The problem could only be solved with the rotating permutation; it appeared for all tested bin and layer numbers with the flipping permutation.

Concluding I have to remark how incredible the samples look, especially in then thinly populated regions of the distribution. It was just recently shown by Villaescusa-Navarro et al. [12] that NNs can offer to reveal deeper insights into physics as long as they offer trustworthy results. A strong NN can therefore lead to groundbreaking new discoveries in giant datasets of surveys like GAIA.

Table 1: Performances of classifiers from different architectures. The order is given by the area under the ROC curve. The rates come from the test dataset with 0.56 % RR Lyrae stars. The left columns show values for a threshold that was optimized for the highest percentage of total correct classifications. The right columns show values for a threshold that was optimized for a highest ratio of tpr to fpr, i.e. the cleanest classifier. The data given here are tp and fp, the total fraction of data that is classified true positive and false positive because they are better comparable with the total correctly classified percentage

| | max. correct | | | cleanest classification | | |
|---|---|---|---|---|---|---|
| Architecture | correct (total) / % | tp / % | fp / % | correct (total) / % | tp / % | fp / % |
| NSF-R 32L 32B | 99.79 | 0.43 | 0.09 | 99.50 | 0.05 | 0.00 |
| MAF-F 4L | 99.77 | 0.38 | 0.04 | 99.58 | 0.14 | 0.00 |
| NSF-R 4L 64B | 99.70 | 0.43 | 0.17 | 99.51 | 0.06 | 0.00 |
| NSF-F 16L 16B | 99.71 | 0.33 | 0.06 | 99.51 | 0.06 | 0.00 |
| NSF-R 8L 8B | 99.73 | 0.42 | 0.13 | 99.53 | 0.09 | 0.00 |
| NSF-R 8L 64B | 99.67 | 0.35 | 0.13 | 99.48 | 0.04 | 0.00 |
| NSF-F 16L 8B | 99.75 | 0.39 | 0.08 | 99.51 | 0.06 | 0.00 |
| NSF-R 4L 128B | 99.74 | 0.37 | 0.06 | 99.53 | 0.09 | 0.00 |
| NSF-F 4L 64B | 99.76 | 0.42 | 0.10 | 99.51 | 0.06 | 0.00 |
| NSF-R 16L 16B | 99.74 | 0.37 | 0.06 | 99.55 | 0.11 | 0.00 |
| NSF-F 8L 8B | 99.74 | 0.34 | 0.04 | 99.51 | 0.06 | 0.00 |
| NSF-F 8L 16B | 99.75 | 0.39 | 0.08 | 99.51 | 0.06 | 0.00 |
| Real NVP | 97.99 | 0.18 | 1.63 | 98.31 | 0.04 | 1.17 |
| NSF-R 64L 8B | 99.44 | 0.00 | 0.56 | 99.44 | 0.00 | 0.56 |
| MAF-R 8L | 93.98 | 0.01 | 5.48 | 93.98 | 0.01 | 5.48 |

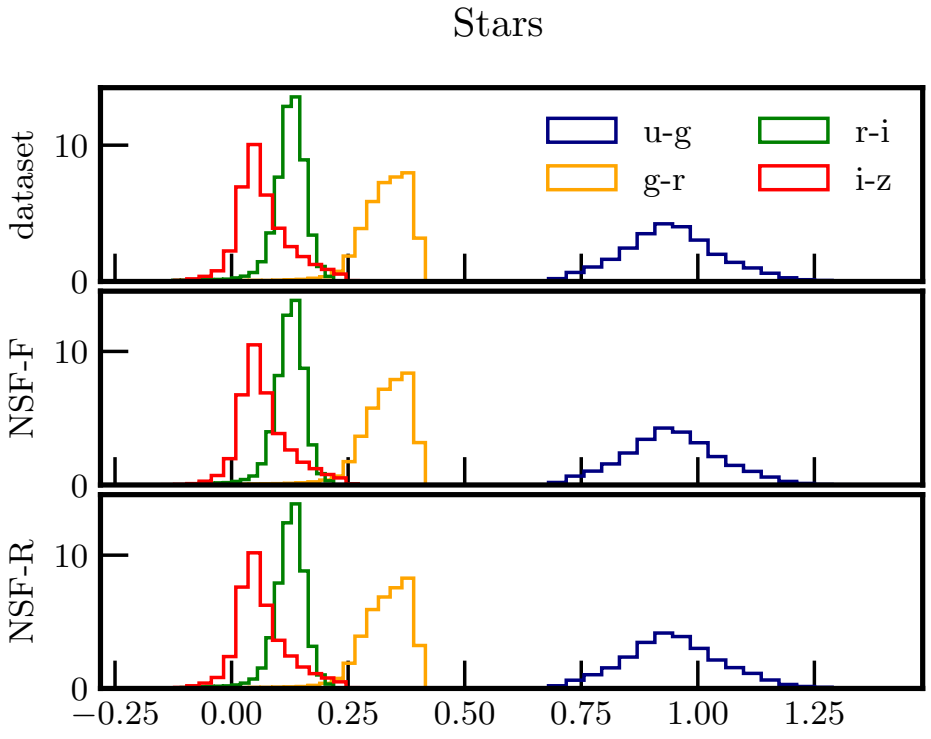


Figure 5: Histograms of the star samples from the 4 layer and 64 bin NSF compared to the dataset and by the permutation pattern

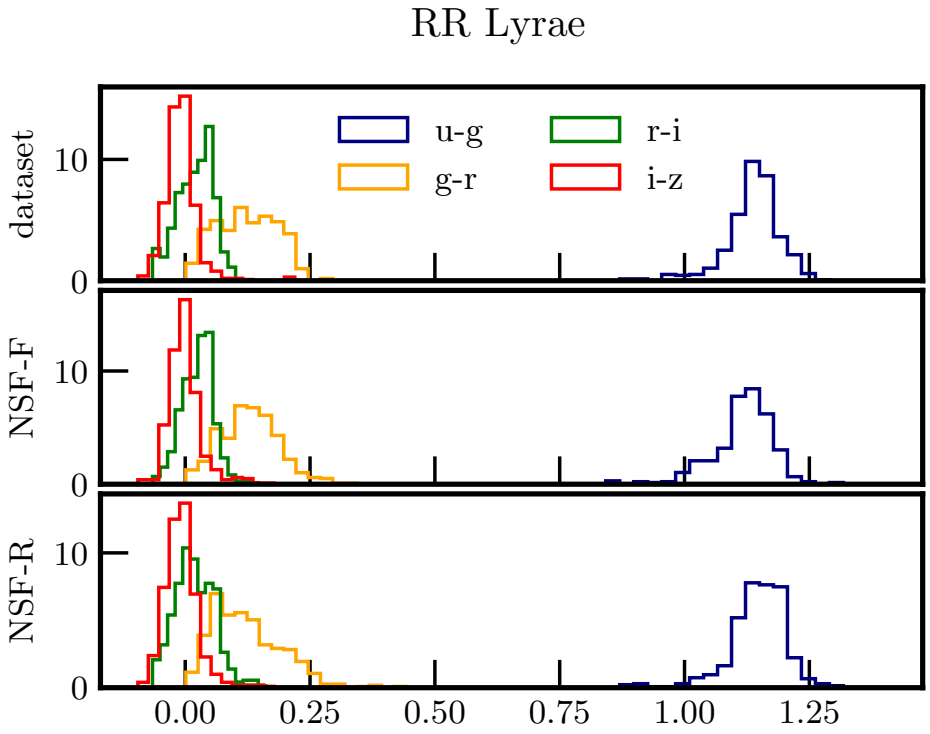


Figure 6: Histograms of the RR Lyrae samples from the 4 layer and 64 bin NSF compared to the dataset and by the permutation pattern

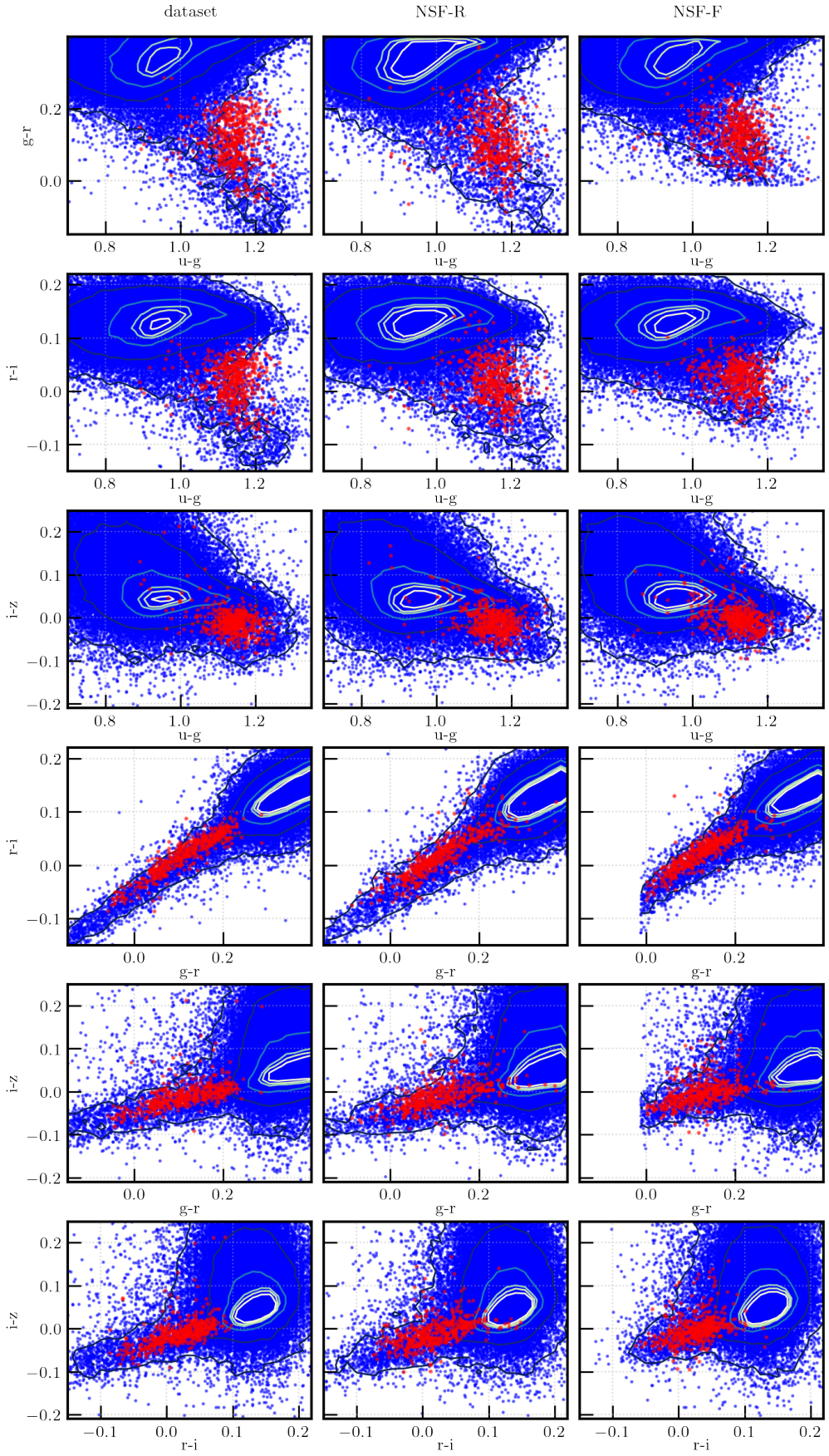


Figure 7: Two-dimensional data samples from the 4 layer and 64 bin NSF with both permutation patterns compared to the original dataset. The contours are used to give more insight to the distribution of the stars