Advanced Methods of Data Analysis: Normalizing Flows

Leonhard Moske
(Dated: August 23, 2022)

In this paper, a recent method named normalizing flows is utilized to construct density estimators for the purpose of classification. Normalizing flows are a class of transformation, which can be trained with gradient descend to estimate the distribution of data or to generate data in a toy model.

In this paper, the expressiveness of fully connected neural networks is used in a class of transformation known as normalizing flows in order to construct density estimator. Further these estimators are utilized to classify stars.

more explizit

I. INTRODUCTION

Normalizing flows is a powerful method that utilizes the transformation of random variables for either density estimation or for generative sampling.

To demonstrate the functionality of normalizing flows we investigate the *AstroML RR-Lyrae variable stars data*-dataset which contains a record of *RR-Lyrae variable stars*. These are periodic variable stars, found in globular cluster, that serve as standard candles to measure extra galactic distances. The data was taken at the *Sloan Digital Sky Survey (SDSS)*. The features provided are four intensities at different wavelength filters corresponding to the photometric system. The four features are: $\{u-g, g-r, r-i, i-z\}$, where u is at 355 nm, g at 468 nm, r at 616 nm, i at 748 nm and z at 893 nm, Figure 1. The dataset contains background stars further named background and the *RR-Lyrae variable stars*, which we are going to call signal. The number of background entries is 92 658 and of the signal 483. In figure 2 we see the features of signal and background and we can see that the features of the signal are grouped within the range of all feature values, making a classification easier. The *Kolmogorov-Smirnov*-test validates this, because all features have scores higher than 0.75, see table I.

citation

cite astroml

cite the dataset https://iopscience.iop.org/article/10.1088/0004-637X/708/1/717/meta

https://www.sdss.org/instruments/camera/

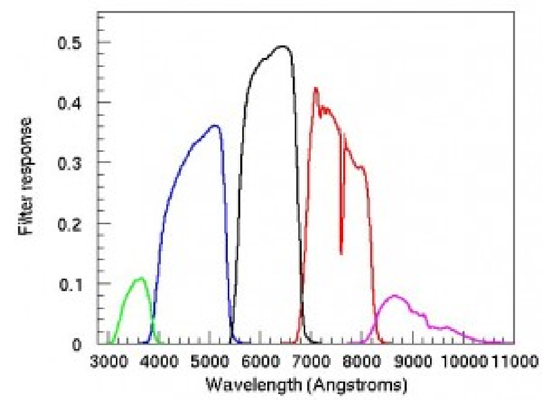

FIG. 1: Filter response to different wavelengths of the *SDSS*-camera. Form left to right the filters are: u, g, r, i and z.

II. THEORY

To classify data into different categories (background and signal) using the density of these categories we have to evaluate the probability density distributions at the data and compare the result, i.e. choose the category with the highest probability. In this case we have only two categories (background named 0, signal named 1), thus we can use a test statistic like:

feature	score
$u-g$	0.80
$g-r$	0.94
$r-i$	0.91
$i-z$	0.76

TABLE I: Kolmogorov-Smirnov score of the four features

$$t(\text{data}) = \ln\left(\frac{p(0|\text{data})}{p(1|\text{data})}\right)$$

We should classify the data as background if $t > 0$ and as signal if $t < 0$. In practice this cut has to be chosen with respect to the validation of the classifiers.

So we have to construct an algorithm that allows us to evaluate the probability density of both categories.

A. functions of random variables

To estimate the density distribution we make use of the formula of transformations of random variables, which lets us connect a simple distribution that we can evaluate and the distribution of the data. This allows us to approximate the evaluation of the data density at points, i.e. new data.

Let z be a random variable distributed as $r(z)$ then a random variable $x = f(z|\theta)$, where f is a invertible and differentiable function with parameters θ, is distributed

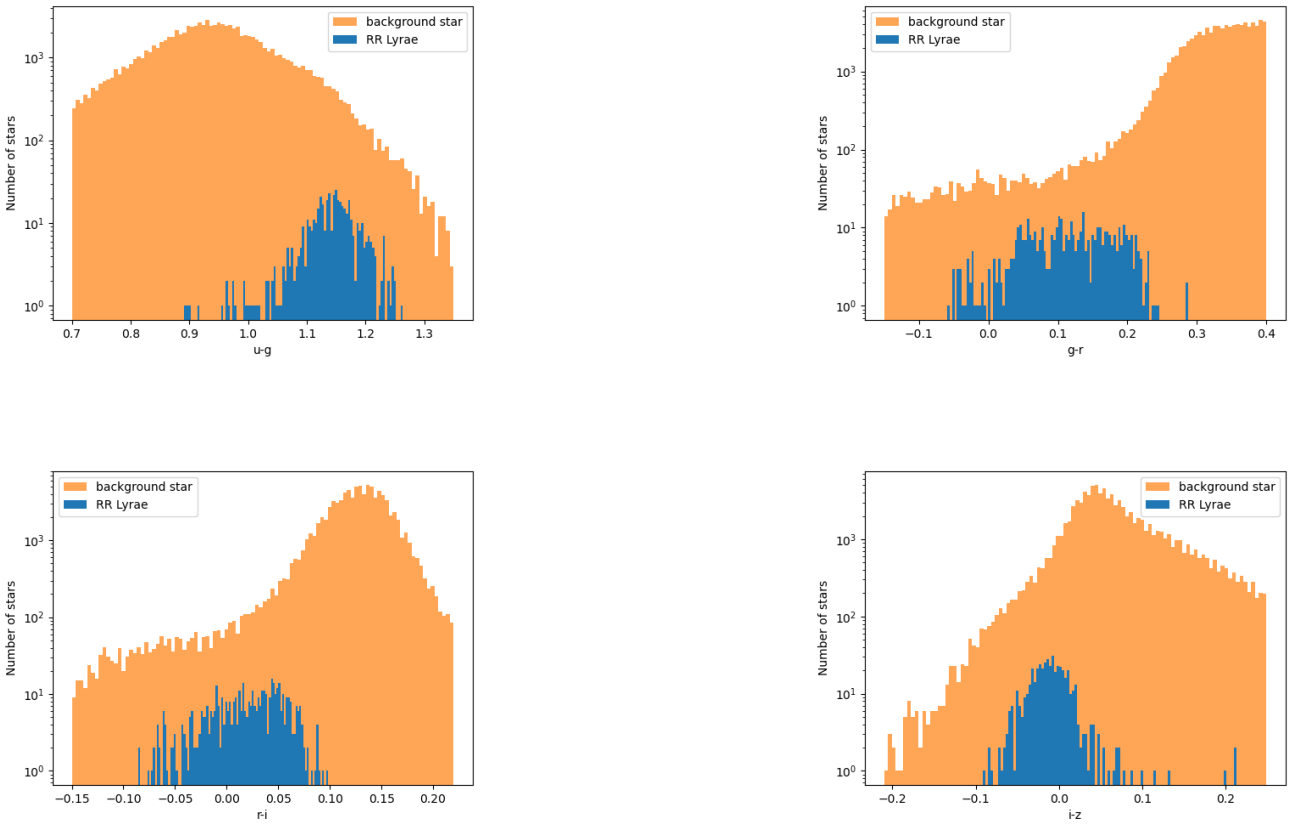

FIG. 2: histograms of the features with the data split into background and signal

as $q(x)$ with:

$$q(x) = r(z)\left|\frac{\mathrm{d}z}{\mathrm{d}x}\right| = r(z)\left|\mathrm{det}J_f(z)\right|^{-1}$$

We call $r(z)$ the base distribution and $q(x)$ the target distribution.

To get a density estimation of new data x' with some parameters we would vary θ until $q(x)$ is close to the target distribution of the data $p(x)$, then we can compute $r(f^{-1}(x'|\theta))\left|\mathrm{det}J_f(x'|\theta)\right|$ which is the estimate for the probability of the data. To do this we have to be able to compute the inverse transformation, its Jacobian determinant and evaluate the base distribution. As the base distribution we choose a multidimensional normal distribution. The dimension of this distribution is the number of features since the flow f has to be injective.

citation

B. parameterize the transformation

To achieve the needed expressiveness of the transformation we construct it by composing smaller so called coupling layers, see also figure 3:

$$\begin{aligned} f &= f_0 \circ f_1 \circ ... \circ f_K \qquad k = 0, 1...K \\ z_k &= f_k(z_{k-1}) \\ z_K &= x \end{aligned}$$

We achieve invertibility and differentiability of f if all coupling layers f_k are differentiable and invertible. Also the Jacobian determinant is calculated as:

$$\begin{aligned} \ln\left|\mathrm{det}J_{f^{-1}}(x)\right| &= \ln\left|\prod_{k=0}^{K}\mathrm{det}J_{f_k^{-1}}(x_{k-1})\right| \\ &= \sum_{k=0}^{K}\ln\left|\mathrm{det}J_{f_k^{-1}}(x_{k-1})\right| \end{aligned}$$

Also because the the coupling layers have to be invertible they have to be injective, having the same number of input as output dimensions.

In order to achieve inveribility of f_k we use a method called *freezing* where some variables are changed by the current coupling layer and others are not.

$$\begin{aligned} x_{i<j} &\to g(x_{i<j}, s(x_{i\geq j})) = x'_{i<j} \\ x_{i\geq j} &\to x_{i\geq j} = x'_{i\geq j} \end{aligned}$$

where g has to be an invertible function, but s does not have to be invertible. We will use artificial neural nets (ANN) to give these transformations the needed expressiveness. The parameter j is dependent on the explicit parametrization of f_k. Between the coupling layers the features are permuted so that all are being transformed or have influence on the transformation.

The inverse of this is then

$$x'_{i<j} \to g^{-1}(x'_{i<j}, s(x'_{i\geq j})) = x_{i<j}$$
$$x'_{i\geq j} \to x'_{i\geq j} = x_{i\geq j}$$

Because of this *freezing* the Jacobian determinant has a sparse form and can be easily computed, but is dependent on the explicit implementation.

citation

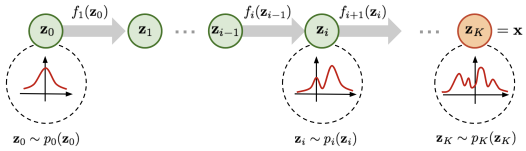

FIG. 3: Schemata of "flow" from the base distribution $r(z)$ to $q(x)$

citation picture URL in comment

C. training the transformation

In order to receive a distribution $q(x)$ that represents the target distribution $p(x)$ closely we have to vary the parameters θ of the transformation f. One can use gradient descent with a divergence as a loss function. In this implementation the Kullback-Leibler (KL) divergence is used as one of the most popular.

In this case where we have samples of the target distribution it is suitable to work with the forward KL divergence between $p(x)$ and $q(x|\theta)$:

$$\begin{aligned}\mathcal{L}(\theta) &= D_{KL}\left[p(x)\|q(x|\theta)\right] \\ &= -\mathbb{E}_{p(x)}\left[\ln(q(x|\theta))\right] + \text{const} \\ &\approx -\frac{1}{N}\sum_{n=1}^{N} \ln(r(f^{-1}(x_n|\theta))) + \ln\left|\det J_{f^{-1}}(x_n|\theta)\right| + \text{const}\end{aligned}$$

where the x_n are the sampled target data. Thus we have to be able to compute f^{-1}, its jacobian determinant and evaluate $r(z)$ and since we want do use gradient descend we need to differentiate through them.

citation

III. NORMALIZING FLOW CATEGORIES

The two normalizing flow parameterizations we used are a Masked Autoencoder (**MADE**) , provided by the tensorflow library and a Real NVP . The transformation of the Real NVP is:

citation

citation

Masked autoencoder

bold

$$x_{0\ldots\frac{d-1}{2}} \to \alpha \cdot x_{0\ldots\frac{d-1}{2}} + \beta$$
$$x_{\frac{d-1}{2}\ldots d-1} \to x_{\frac{d-1}{2}\ldots d-1}$$
$$\alpha = s(x_{\frac{d-1}{2}\ldots d-1}) \qquad \beta = s(x_{\frac{d-1}{2}\ldots d-1})$$

where d is the number of features and s is a fully connected neural net with $ReLU$ activation functions. The last layer of the scale α is activated with tanh while the shift β is linear activated. Then the Jacobian determinant is:

$$\ln\left|\det J_{f_k^{-1}}(x_{k-1})\right| = \sum_{i=0}^{\frac{d}{2}} \ln(\alpha_i)$$

Between these transformations we apply the permutation (0123) of the features.

IV. METHODS

We use the tensorflow library to build the normalizing flows. Especially essential are the bijector and distribution classes of the probability module. We split the data into signal and background, and further into training, testing and validation subsets. These contain 80%, 10% and 10% of the data. The training data is divided into batches. As the base distribution $r(z)$ we choose a multidimensional normal distribution. The transformation is implemented as a *tensorflow Chain* of *tensorflow bijectors*. These bijectors are the coupling layers and the permutation from section II B. All bijectors have methods for computing the forward transformation, inverse of the transformation and the logarithmic Jacobian determinate.

For training the flow we iterate over all batches and apply the gradient of the loglikelihood with the adams optimizer using *tensorflow GradientTape*. The learning rate decays from a set initial value to a set final value with a polynomial decay with power $\frac{1}{2}$. We train the models until the relative change of the loss function on the testing data is bellow 1×10^{-6}

citation luRi

For evaluating the validity of the model we calculate the test statistic t from section II, also we calculate the fraction of correctly classified data from the validation data given some cut and we look at the ROC-Curve of the test statistic.

Rewrite Code no LURI

comments code

V. RESULTS

We first trained the models on default parameters, listed in table II. Since the number of background entries and signal entries differ greatly we used different batchsizes. The cut specifies at which value of the test statistic t we classify a data point as background or signal. Layers determines how many coupling layers and permutations make up the model. There are also model specific parameters which control the shape of the **ANN**s.

parameter	value
batchsize 0	1000
batchsize 1	100
cut	0
start learningrate	1×10^{-3}
final learningrate	1×10^{-4}
layers	8

MADE		Real NVP	
nodes per hidden layer	200	nodes per hidden layer	100
number of hidden layers	2	number of hidden layers	4

TABLE II: Default parameters for the normalizing flow models and training

	Real NVP	MADE
correctly classified	0.9870	0.9831
false positive	0.0126	0.0166
false negative	0.0003	0.0002

TABLE III: Results of the classification for Real NVP and MADE.

Both models result in histograms of the test statistic as in 4 We can see that both models deliver a negative response for the signal. For the background we observe that most of the data has a positive test statistic t, thus a classification of signal and background with a cut at $t = 0$ is the best cut to minimize false negative classification. But we also notice that then some background data is classified as signal, making a 100 % accurate classification in this manner unachievable. With a cut of 0 the results of the classification are shown in table III, with

$$\text{correctly classified} = \frac{\#\text{true positive} + \#\text{true negative}}{\#\text{all}}$$

$$\text{false positive} = \frac{\#\text{false positive}}{\#\text{all}}$$

$$\text{false negative} = \frac{\#\text{false negative}}{\#\text{all}}$$

.

For different cuts we look at the ROC-curves of both models

ROC beschr

Loss beschr

Summary

conclusion

VI. SUMMARY

VII. CONCLUSION

what left to do

quellen raus-suchen bzw.

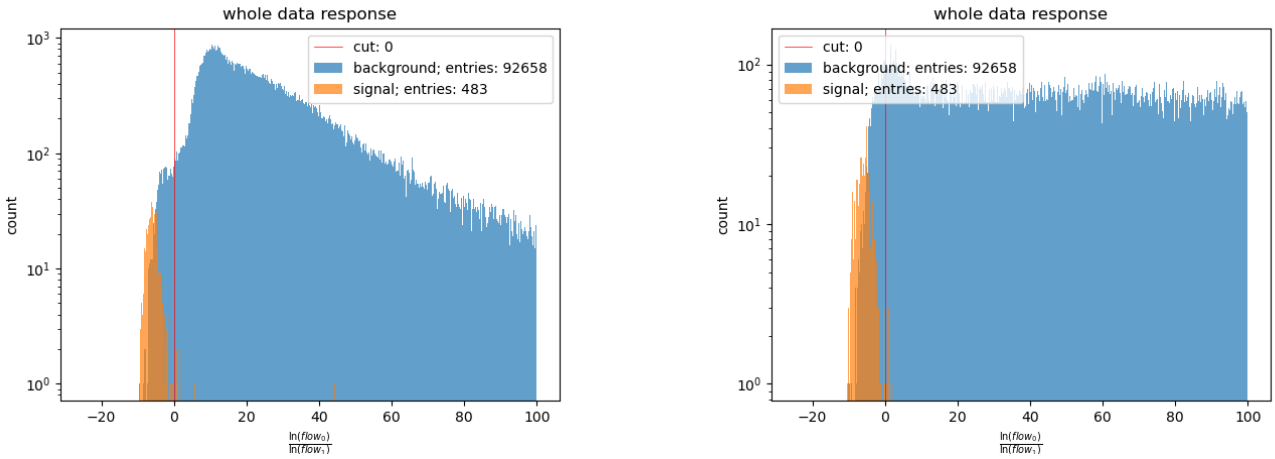

FIG. 4: histograms of the test statistic t for (left) Real NVP (right) MADE

FIG. 5: ROC-Curve of the Real NVP (left) and MADE (right)

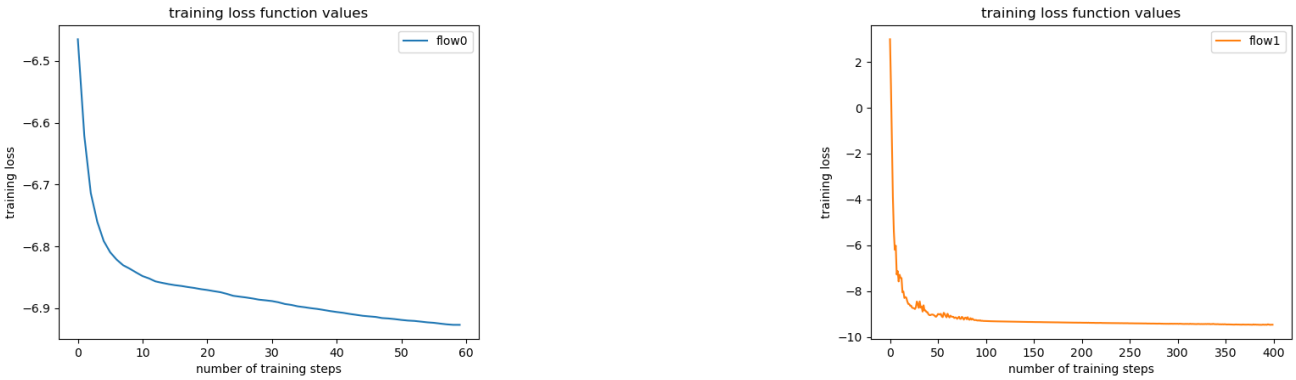

FIG. 6: loss function over time on the training data