UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA INTERINSTITUCIONAL DE PÓS-GRADUAÇÃO EM ESTATÍSTICA UFSCar-USP

Marco Henrique de Almeida Inácio

Conditional independence testing, two sample comparison and density estimation using neural networks

Tese apresentada ao Departamento de Estatística – Des/UFSCar e ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre ou Doutor em Estatística - Programa Interinstitucional de Pós-Graduação em Estatística UFSCar-USP.

Orientador: Prof. Dr. Rafael Izbicki

São Carlos
Agosto de 2020

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia
Programa Interinstitucional de Pós-Graduação em Estatística

Folha de Aprovação

Defesa de Tese de Doutorado do candidato Marco Henrique de Almeida Inácio, realizada em 03/08/2020.

Comissão Julgadora:

Prof. Dr. Rafael Izbicki (UFSCar)

Prof. Dr. Francisco Aparecido Rodrigues (USP)

Prof. Dr. Diego Furtado Silva (UFSCar)

Prof. Dr. Bálint Gyires-Tóth (BME)

Prof. Dr. Anderson Luiz Ara Souza (UFBA)

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

This work is dedicated to the most Holy Trinity and the Θεοτόκος.

ACKNOWLEDGEMENTS

First of all, I would like to say thanks to my parents Nivaldo and Cláudia and to my brother João Pedro. Then to my advisor Rafael Izbicki, and professors Bálint Gyires-Tóth, Rafael Stern and Luis Ernesto Bueno Salasar for their amazing help and dedication to this work.

And then professors of the evaluation board (Anderson Luiz Ara Souza, Diego Furtado Silva and Francisco Aparecido Rodrigues) and the DE/UFSCar graduation secretaries Isabel Araujo and Celius Martinez.

I would also like to thank the financial support of CAPES (this study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001), of the Erasmus+ program and of the BME-Artificial Intelligence FIKP grant of Ministry of Human Resources (BME FIKP-MI/SC). Finally, I would also like to thank the professors and employees of University of São Paulo, University of São Carlos and of the Budapest University of Technology and Economics.

“ Ἅγιος ὁ Θεός, Ἅγιος ἰσχυρός, Ἅγιος ἀθάνατος, ἐλέησον ἡμᾶς. ”
“ Sanctus Deus, Sanctus Fortis, Sanctus Immortalis, miserere nobis. ”
“ Szent Isten, szent Erős, szent Halhatatlan, irgalmazz nekünk. ”
“ Santo Deus, Santo Poderoso, Santo Imortal, tende piedade de nós. ”
“ Holy God, Holy Mighty, Holy Immortal, have mercy on us. ”
(Trisagion)

RESUMO

INACIO, M. H. A. **Estimação de densidades, comparação de amostras e medidas de importância usando redes neurais**. 2020. 113 p. Tese (Doutorado em Estatística – Programa Interinstitucional de Pós-Graduação em Estatística) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2020.

Dada a grande quantidade de dados disponíveis nos dias de hoje e o rápido aumento da capacidade de processamento computacional, o campo de aprendizado de máquina e a assim chamada modelagem algorítmica tem visto um grande surto de popularidade e aplicabilidade. Uma das ferramentas que atraíram grande popularidade são as redes neurais artificiais dada, entre outras coisas, sua versatilidade, habilidade de capturar relações complexas e sua escalabilidade computacional. Assim sendo, neste trabalho aplicamos estas ferramentas de aprendizado de máquina em três problemas importantes da Estatística: comparação de populações, teste de independência condicional e estimação de densidades condicionais.

Palavras-chave: redes neurais artificiais, estimação de densidade condicional, teste de independência condicional, comparação de populações, aprendizado de máquina.

ABSTRACT

INACIO, M. H. A. **Conditional independence testing, two sample comparison and density estimation using neural networks**. 2020. 113 p. Tese (Doutorado em Estatística – Programa Interinstitucional de Pós-Graduação em Estatística) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2020.

Given the vast amount of data available nowadays and the rapid increase of computational processing power, the field of machine learning and the so called algorithmic modeling have seen a recent surge in its popularity and applicability. One of the tools which has attracted great popularity is artificial neural networks due, to among other things, their versatility, ability to capture complex relations and computational scalability. In this work, we therefore apply such machine learning tools into three important problems of Statistics: two-sample comparison, conditional independence testing and conditional density estimation.

Keywords: artificial neural networks, conditional density estimation, conditional independence testing, two-sample comparison, machine learning.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SOURCE CODES

LIST OF TABLES

CONTENTS

CHAPTER 1
INTRODUCTION

Given, among other things, the vast amount of data available nowadays and the rapid increase of computational processing power, the field of machine learning and the so called algorithmic modeling have seen a recent surge in its popularity and applicability. One of the tools which has attracted great popularity is artificial neural networks due, to among other things, their versatility, ability to capture complex relations and computational scalability. In this thesis, we therefore apply such machine learning tools into three traditional problems of Statistics: two-sample comparison, conditional independence testing and conditional density estimation.

In chapter 2, we present an introduction to dense neural networks, its properties, as well as the state-of-the-art methods of optimization and regularization; we also present a description of the deep learning framework used.

In chapter 3, we present an introduction to Monte Carlo simulation studies and to the *sstudy* package, with a short statistical description of the simulation study procedure, as well as some examples of application using the package. Monte Carlo simulation studies (which can be defined as the procedure of estimating and comparing properties of estimators by averaging over many replications given a true distribution) are a possible way to access the performance of new machine learning/statistical estimators. The *sstudy* package, on the other hand, is designed to simplify their preparation and execution using SQL database engines. Both simulations studies and this package will be vital to carry out the work in the succeeding chapters. An article with content related to this chapter has been published by Inácio (2020).

In chapter 4, we present a novel nonparametric approach to assess the dissimilarity between two high-dimensional datasets using variational autoencoders (VAE) (KINGMA; WELLING, 2014). We show how our approach can be used to visually assess how far apart datasets are from each other via a boxplot of their distances and additionally, provide a way of interpreting the scale of these distances by using the distance between known distributions as a baseline. We also show how a formal permutation-based hypothesis testing can be derived within our framework. This chapter solves two gaps in literature, first a general one which is lack of methods for population comparison on high dimensional data using state of the art techniques. The second, which is more general in scope, is the urging necessity to bring the novel methods of machine learning to revisit and solve decades old classical problems of Statistics. The proposed methods could be used for various tasks in the machine learning pipeline, including: distribution shift detection and measurement, self-supervised clustering and anomaly detection. An article with content related to this chapter has been published by Inácio, Izbicki and Gyires-Tóth (2020).

In chapter 5, we present the problem of importance measure and testing and our proposed method of addressing such problem. We also propose a simulation study to investigate the properties of such method and present the results of such study on simulated datasets. Conditional independence testing is required by many methods in machine learning and statistics, including Bayesian networks, time series, causal inference and feature selection. However, it is not possible to design conditional independence tests that are powerful against all points in the alternative hypothesis (SHAH; PETERS, 2018). This issue is partially addressed by making assumptions about the data distribution. This chapter develops conditional independence testing to evaluate the usefulness of features on a prediction problem. Our goal in this chapter then is to test whether $\mathbf{X}_A$ is independent of the label, Y, given $\mathbf{X}_O$, that is, whether $H_0 : \mathbf{X}_A \perp Y | \mathbf{X}_O$ holds. We show that the p-values obtained by our approach are proper, and that our hypothesis test has larger power than competing approaches under a variety of settings. An article with content related to this chapter has been published by Inácio, Izbicki and Stern (2019).

In chapter 6, we present an introduction to the conditional density estimation problem and to Fourier Series. We also present our proposed method of conditional density estimation as well an study of the performance of such method using real world datasets with social, IT and astronomy related data. An article with content related to this chapter has been published by Inácio and Izbicki (2018).

Finally, in chapter 7, we conclude the thesis with final remarks.

CHAPTER

2

NEURAL NETWORKS: INTRODUCTION, TUNING AND OVERFITTING

In this chapter, we present an introduction to dense neural networks, its properties, as well as the state-of-the-art methods of optimization and regularization; we also present a description of the deep learning framework used.

Thought-out this chapter, we assume that the observations (instances, a row of the dataset) of the dataset are independent and identically distributed.

2.1 Dense neural networks

In their simplest case, with dense (also called linear or fully connected) connections, for $l > 0$, each node x_k^{l+1} is linear combination of all the nodes of the previous layer plus an intercept (also called bias) followed by applying an activation function $\mathbb{F}^l$.

In Figure 1, we show a graphical representation of a neural network with 2 inputs, 3 outputs and 2 hidden layers and with weights and activation functions omitted for simplicity. On the other hand, in Figure 2, we show a detail representation of part of the same neural network, including the weights (in green) and operations and activation function (in yellow).

Despite being very powerful, neural network, however, also exhibit their own set of problems, two of them being overfitting due to non-linearity[1] and difficulties in convergence to the global minimum due to the particular non-convex structure of the problem (see Reddi, Kale and Kumar (2018), for instance). In the following sections we describe among other things, methods to work around such problems.

[1] In general, the larger the class of possible functions that a model can fit, the more it will suffer from overfitting. See Hastie, Tibshirani and Friedman (2001), for instance.

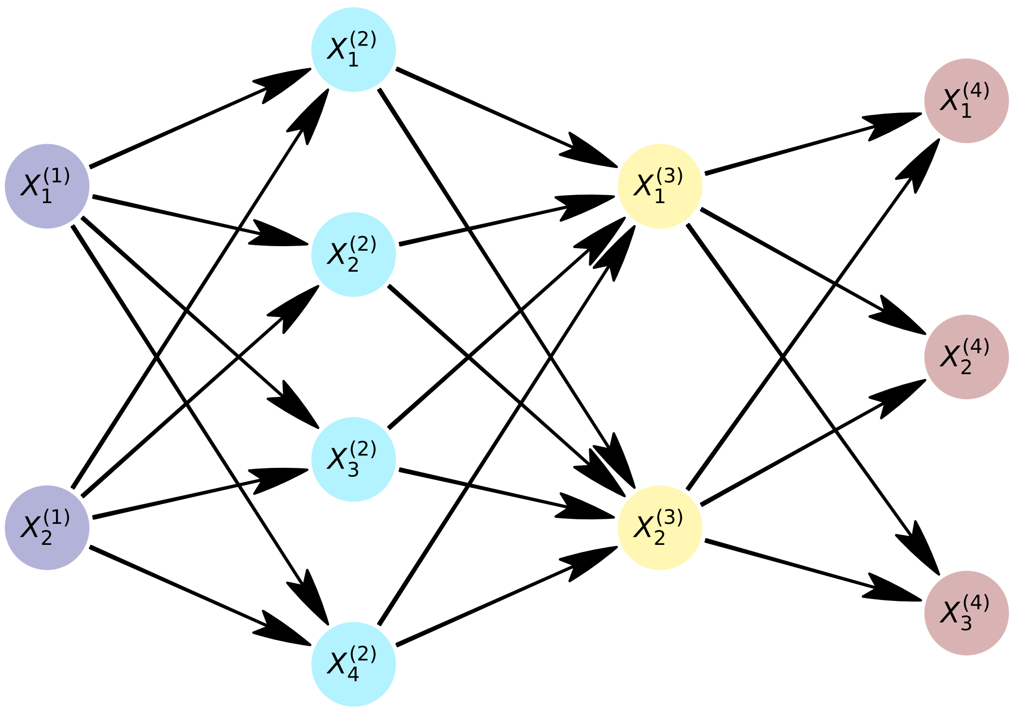


Figure 1 – Example of a neural network with 2 inputs, 3 outputs and 2 hidden layers. The weights and activation functions were omitted for simplicity.

2.2 Loss, SGD, learning rate, batch size and optimizers

Given a neural network, a loss function L and some input data, the outputs can be used as predicted values and then one can use them to calculate a loss. L can be, for example: the mean squared error between predicted values and target values (analogous to the one calculated in a least squares regression procedure).

A simple way to update the weights of the neural network to minimize the loss in a training dataset is by using the gradient descendant (GD) method. Basically, it consists of using the whole dataset to calculate to gradient the loss with respect to the neural network weights and then update the weight in the direction of the gradients with such update weighted by a learning rate ε:

$$\beta \leftarrow \beta + \frac{\mathrm{d}L}{\mathrm{d}\beta}\varepsilon \tag{2.1}$$

Another possibility, the stochastic gradient descendant (SGD). In its simplest form, called online SGD, the gradient is calculated (and the weights are updated) using a single observation at a time. Moreover, in between the GD and the online SGD we have what is called batch SGD which instead uses a part of the dataset (i.e.: a batch) to calculate the gradients and update the weights, this however introduces another tuning parameter

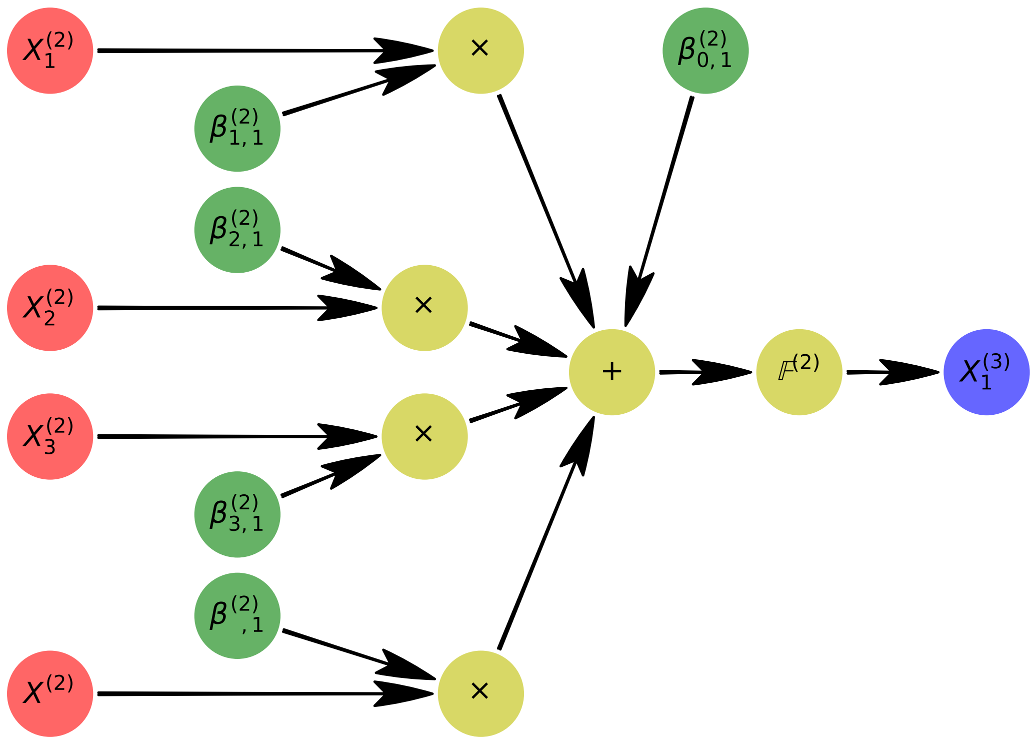


Figure 2 – Detailed representation of part of the neural network in Figure 1, including the weights (in green) and operations and activation function (in yellow).

that affect the trajectory and convergence of the algorithm: the batch size. Note that the general procedure for both SGD forms is to permute the dataset before each epoch (here an epoch means a pass through the dataset).

One of the greatest advantages of using SGD is its ability avoid convergence to a bad local minimum given the randomness introduced by using only part of the dataset on each optimization step (weights update). However this ability is greatly affected by the chosen learning rate (see Bottou (1991), for instance). In this work, in particular, we use the Adam optimizer (KINGMA; BA, 2014) to handle the learning rate of the optimization procedure.

Careful initialization is also of great help in avoiding those problems. Regarding this, we used a method of initialization proposed by Glorot and Bengio (2010a).

2.3 Activation functions

As we have seen, after combining the weighted values of the previous neurons and a bias, we apply an activation function before storing the value in the next neuron. Among other things, these activation function aims to allow the network to model non-linear

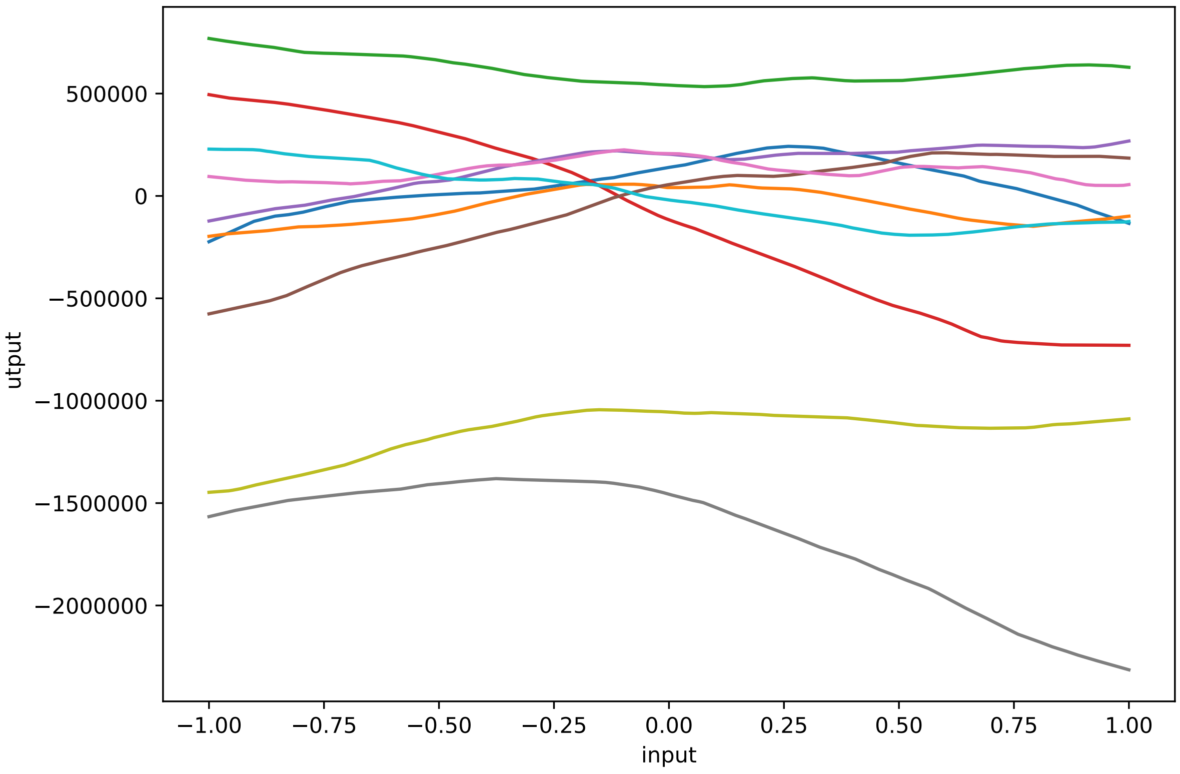


Figure 3 – Input and output of a neural network with a single input neuron, a single output neuron and 3 hidden layers, each of them with 100 neurons where each plotted curve has different weights. All layers, except the last one were activated using ReLU.

functions and provide numerical stability (specially for the gradients[2]). Regarding this, one common possibility is the $\text{ReLU}(x) \equiv \max(0, x)$ activation function.

To illustrate the non-linearity property of it, in Figure 3 we have the input and output of a neural network with a single input neuron, a single output neuron and 3 hidden layers, each of them with 100 neurons where each plotted curve has different weights (randomly chosen). All layers, except the last one were activated using ReLU. Note that without an activation function (i.e.: an identity activation function for all layers), the output would always be a linear function of the input, regardless of the number of hidden layers and the amount of neurons on each layer.

Despite its power and efficiency, ReLU also has its own set of problems, the most important one probably regarding dead neurons, that is, whenever the inputs of the activation function are sufficient small, it will have gradient zero and depending on the circumstances, will be unable to recover from such state, thus the neuron is "dead". Among the proposed solutions to this potential problem we could mention ELU (DJORK-ARNÉ; UNTERTHINER; HOCHREITER, 2016) and Randomized leaky ReLU (XU *et al.*, 2015).

[2] Some activation functions, specially prior to the "advent" of ReLU (HAHNLOSER *et al.*, 2000), are prone to a problem called vanishing gradient, where the gradient computationally evaluates to 0 for large input values given a limited float point precision.

2.4 Regularization and early stopping

An intuitive and common way to avoid overfitting on neural networks, analogous to the usage of LASSO and ridge regression techniques (for linear regression problems), is by using a L2 or L1 regularization (also called penalization) over the weights of the neural network . In case of L2 regularization, it consists of increment the calculated loss by:

$$\varpi \sum_{i,j,l} (\beta_{i,j}^{(l)})^2 \tag{2.2}$$

where $\varpi > 0$ is a tuning parameter. An L1 regularization works in similar fashion, but with absolute value instead of square in Equation 2.2.

Another important technique to avoid overfitting is by early stopping: it consist of separating the dataset in train and validation sets (e.g.: 90%/10% split), using the train set in the optimization (training) procedure of the network and after each epoch, evaluating the error in the validation set and stopping the procedure after a ρ number of epochs without improvement in the validation error. Where ρ is generally called “patience”.

2.5 Batch normalization

In this work, we also take extensively advantage of batch normalization, as proposed by Ioffe and Szegedy (2015), which is a post-activation normalization procedure that aims to stabilize the internal covariate shift of neural networks, that is, the (potentially disturbing) influence that updating a layer of the network causes over the subsequent layers during the training procedure.

2.6 Dropout

Dropout is technique proposed by Srivastava *et al.* (2014) that consist at randomly setting some of the neurons to zero (drooping) during the training procedure. This is aimed at reducing co-adaptation of neurons, a problem that arises specially in small datasets in which a neuron is only helpful in the context of various other neurons and that causes bad performance outside the training dataset (and thus can be considered an overfitting problem). The intuition here is that due to the fact the some of the neurons are dropped randomly, a neuron cannot rely on any other specific neuron or part of the network and therefore co-adaptation is greatly reduced.

As an illustration of the procedure, in Figure 4, we present a neural network with the same structure as in Figure 1, but with two neurons dropped out.

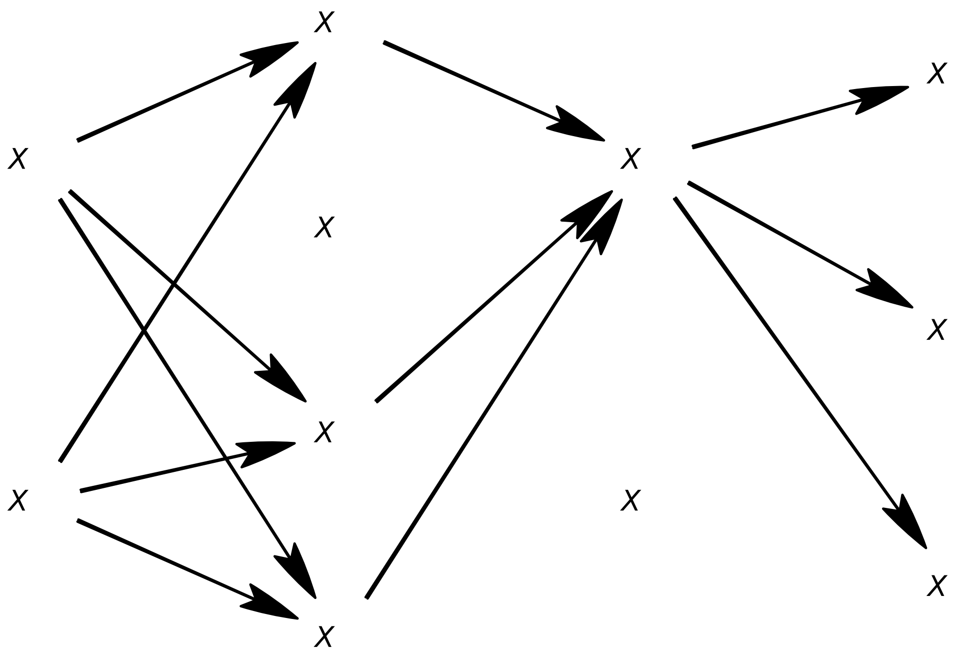


Figure 4 – A neural network with the same structure as in Figure 1, but with two neurons dropped out.

In this work, we used the alpha dropout proposed by Klambauer *et al.* (2017)[3] which is purported at preserving batch normalization transformation (the dropout is applied after the normalization).

2.7 PyTorch, automatic differentiation and GPU

Most deep learning frameworks like TensorFlow and Theano work with symbolic differentiation and therefore it is necessary to declare the model structure before actually supplying any data and then ask the framework to compile the model (having the gradients calculated symbolically), after which point, the model will be good to go as long it does not change.

While those have advantages like possible algebraic simplifications when working out the derivatives, they also come at some price: they impose a necessary “recompilation” of the model if it changes, they make code less intuitive to writing and more difficult to debug, are potentially more restrictive on model characteristics.

[3] This paper also proposed a new activation function called SELU which aims at doing batch normalization using the activation function.

On the other hand, the framework of choice for this work, called PyTorch works with reverse-mode automatic differentiation which consists of creating the chain of differentiation (on a “tape”) on-the-fly. That is, after the last operation is done (in our case, that’s the calculation of the loss), the chain of operations is back-propagated (RUMELHART; HINTON; WILLIAMS, 1986) (i.e.: running the “tape” backwards) and the gradients of the parameters of interest are calculated using chain rule.

Frameworks which works using symbolic differentiation are often called static while the ones that use automatic differentiation are called dynamic. Regardless of this, most (if not all) of those deep learning framework have two common characteristic that are worth emphasizing: they allow one to use its differentiation facilities to work with problems other than deep learning, neural networks or optimization (e.g.: Markov Chain Monte Carlo and Bayesian inference) and they natively support GPU acceleration (generally, using Nvidia CUDA).

The reason GPU acceleration is a common denominator over the deep learning frameworks is due to the fact that neural networks are strongly parallelizable problems and this make them well suited for GPU acceleration. This thus explain, at least in part, their recent surge in popularity given the scalability properties of such methods to big datasets, which on the other hand, are getting increasingly common and important.

2.8 Conclusion

In this chapter, we have presented an introduction to dense neural networks, its properties, as well as a description of the deep learning framework to be used in this work.

As a final remarks for this chapter, we notice that neural networks are not limited to dense/linear connection types that we described here. Among those, we could mention in particular first convolutional neural networks (FUKUSHIMA, 2007) which have given incredible breakthrough in the area of image recognition but also recurrent neural networks (RUMELHART; HINTON; WILLIAMS, 1986) which extend dense neura network to sequential data (possibly of variable size) and have successfully been applied to problems of natural language processing, among others.

CHAPTER 3

MONTE CARLO SIMULATION STUDIES ON PYTHON USING THE SSTUDY PACKAGE WITH SQL DATABASES AS STORAGE

In this chapter, we present an introduction to simulation studies and to the *sstudy* package, both simulations studies and this package will be vital to carry out the working in the succeeding chapters.

3.1 Introduction

One important aspect of proposing new machine learning/statistical estimators and methods is the performance test phrase. A possible way to access such performance is by Monte Carlo simulation studies, which can be defined as the procedure of estimating and comparing properties (such as predictive power) of estimators (and other statistics) by averaging over many replications given a true distribution; i.e.: generating a dataset, then fitting the estimator, calculating and storing the predictive power, and then repeating the procedure many times and finally averaging over the predictive powers across repetitions.

In this chapter, we present a Python (ROSSUM, 1995) package called *sstudy* which is designed to simplify their preparation and execution using SQL database engines. We also present a short statistical description of the simulation study procedure, as well as some examples of application using the package.

3.1.1 Related work

While applications of simulation studies are abundant in literature (with a simple search of "Monte Carlo simulation study" on *Google Scholar* yielding up to a thousand results), there is also a considerable literature regarding the discussion and meta analysis of Monte Carlo simulation studies themselves. In particular, Morris, White and Crowther (2019) presents a Biostatistics tutorial on the rationale behind using simulation studies, while providing guidance for their design, execution, analysis, reporting and presentation, with special focus on method evaluation.

Burton *et al.* (2006) discuss, in the context of medical Statistics, issues to consider when designing a simulation study; in particular, by exposing how the study will be performed, analysed and reported in details. Moreover, design decisions are discussed such as the procedures for generating datasets and the number of simulations to be performed. The authors also suggest a checklist of important design considerations.

Metcalfe and Thompson (2005) discuss the effect of varying the event generation process (data generating distribution) on simulation studies in the context of evaluation of Statistical methods for the analysis of recurrent events. Four distinct generating distributions (Poisson, mixed Poisson, autoregressive, and Weibull) are used to evaluate a set of distinct statistical estimators and their impact in the results is analysed. The authors conclude that the event generation process impact the quality of the estimator and that, therefore, multiple generation processes should be considered on a study.

Mundform *et al.* (2011) discuss the choice of the number of replications (simulations) to be done for simulation studies with regards to the quality of the simulation, in terms of, for example, type I error rates, power and run time. 22 works in literature were analysed by them and replicated in order to find the minimum number of simulations to be done in order to achieve stable results. The authors concluded that in many cases fewer simulations than the original ones used in the works were needed to produce stable estimates of the results, and that, for all works, less than 10000 simulations were sufficient to achieve stable results.

Schaffer and Kim (2007) also discuss the choice of the number of replications to be done on simulation studies, but specifically in the context of control chart analysis. They also conclude than less than 10000 simulations are sufficient to achieve the desired performance in their desired criteria, and moreover, that in many cases less than 5000 were also enough.

Finally, Mooney (1997) provides an extensive book on the subject and presents, among other things, the logic behind Monte Carlo simulation studies, a set of five steps to implement them and discusses their use in social sciences.

3.1.2 Terminology

Given the mixed audience nature of this work, we use the following terms interchangeably in this chapter:

- Train model, fit model to data.
- Dataset, sample.
- Number of instances, sample size.
- Dataset generator, true distribution, data generating function.
- Loss, decision criteria.

3.1.3 Chapter organization

The rest of this chapter is organized as follows: in Section 3.2, we present a short introduction on simulation studies, with a short statistical notation of what is being estimated by them. In Section 3.3, we present a short introduction on *sstudy* package, with part of the source code of an example of basic usage as well as presenting its features and their usage examples available in its documentation. In Section 3.4, we present some examples of applications of simulation studies using the *sstudy* package with results and analysis. The source code for such examples is distributed together with the package. Finally, section 3.5 concludes the chapter.

3.2 On simulation studies:

The process of a simulation study consists of varying some aspects of the data generating function, the estimating model and estimating its performance by averaging over distinct random seeds for the data generator; i.e., estimating:

$$E_{D \in \mathbb{D}_P}[\text{loss}(M_k(D), D)]$$

where

- P are the parameters of the data generating function (e.g.: distribution parameters, number of instances, etc).
- k are model parameters (e.g.: whether you are using a linear regression, a LASSO or a ridge, and if a LASSO/ridge, what is its tuning parameter, etc).

So, in other words, a simulation study is a repetition of the following procedure many times followed by averaging over the results: generate a dataset D from a ground truth distribution $\mathbb{D}_P$, train a model M_k using this dataset and then evaluate the loss.

Note however, that in order to avoid overfitting, one must train and evaluate the loss on distinct partitions of the dataset D[1]. Algorithm 1 summarizes the procedure.

[1] Note that, in some cases, however it is not necessary to generate a test dataset because a

Algorithm 1 – Simulation study procedure

Input: dataset generator $\mathbb{D}_P$, model M_k, number of desired simulations n_sim.
Output: loss over simulations.

```
1: for $i \in \{1, \dots, n\_sim\}$ do
2:     Generate dataset $D_{\text{train}}$ and $D_{\text{test}}$ from $\mathbb{D}_P$.
3:     Train model $M_k$ using $D_{\text{train}}$: i.e. calculate $M_k\left(D_{\text{train}}\right)$.
4:     Evaluate loss $\left(M_k\left(D_{\text{train}}\right), D_{\text{test}}\right)$ and store it on $L_i$
5: end for
6: Return the mean of $L$.
```

3.3 The sstudy package

In this section, we present the basic usage of the package as well as some of its features.

3.3.1 The package

The *sstudy* Python package is a thin layer designed to simplify the preparation and execution of simulation studies using SQL database engines. The package works by randomly selecting a simulation study configuration and checking if this configuration has already seem the user requested number of simulations; then, if that's the case, skip to another randomly selected simulation configuration, otherwise, do the simulation study and store the results on the SQL engine.

Given the independence of each simulation study and the atomicity of SQL engines[2], this procedure can be parallelized (i.e.: one can span multiple simulation study processes), even over multiple machines. Moreover, if the procedure is stopped abruptly, simulation studies previously done will be not affected as they are already stored in SQL database.

Note that this process of "checking" (and then assigning the simulation configuration to be done) in general involves only a few (mili)seconds (provided the communication with the SQL server is fast enough), so that in the general terms, the bottleneck of execution is the simulation studies themselves.

3.3.2 Storage engines

The package is build around the *peewee* Python package which as of the date of this chapter, supports SQLite, PostgreSQL, MySQL and CockroachSQL.

dataset might not be necessary at all in the evaluation phrase; e.g.: if you want to compare the estimated parameter directly to the true model parameter.

[2] Atomicity: this means that either a transaction (e.g.: storing the simulation study results) will happen entirely, or it won't happen at all.

Source code 1 – Part of the database structure file

```
class Result(Model):
    # Data settings
    data_distribution = TextField()
    method = TextField()
    no_instances = DoubleField()

    # Results
    score = DoubleField()
    elapsed_time = DoubleField()
```

For simple project running on a single machine, SQLite is the recommend storage system as it is a self contained SQL engine which is stores the dataset in a single user-defined file and does not require the installation of a SQL server system. Moreover, SQLite files can be opened and explored using GUI tools such as the DB browser SQLite.

For projects running on a multiple machines, PostgreSQL (or MySQL) is the recommend storage system as it is a well supported open source SQL engine, although it requires the a server installation (or renting a pre-installed server provided by a cloud services platform). CockroachDB on the other hand is a distributed SQL system which can be stored on a cluster on machines.

3.3.3 Basic usage

The recommended design of a experiment using the *sstudy* package is by having it separated in 3 files:

- A file for database structure where we declare the variables to be stored in the SQL database and their respective types (see Listing 1).
- A file for running the simulations where we declare the list of parameters to be simulated, as well as the simulation script itself (see Listing 2).
- A file to explore/plot the results which can be exported directly into a *pandas.DataFrame* (see Listing 3).

To see the complete source code of listings 1, 2 and 3, see the examples/basic folder distributed together with the package, which is also available at: <https://gitlab.com/marcoinacio/sstudy/-/tree/master/examples/basic>.

Source code 2 – Part of the simulation execution file

```
to_sample = dict(
    data_distribution = ["complete", "sparse"],
    no_instances = [100, 1000],
    method = ['ols', 'lasso'],
)

def func(
    data_distribution,
    no_instances,
    method,
    ):

    x = (no_instances + 10000, 10)
    x = stats.norm.rvs(0, 2, size=x)
    beta = stats.norm.rvs(0, 2, size=(10, 1))
    eps = (no_instances + 10000, 1)
    eps = stats.norm.rvs(0, 5, size=eps)
    if data_distribution == "complete":
        y = np.matmul(x, beta) + eps
    elif data_distribution == "sparse":
        y = np.matmul(x[:,:5], beta[:5]) + eps
    else:
        raise ValueError

    y_train = y[:no_instances]
    y_test = y[no_instances:]
    x_train = x[:no_instances]
    x_test = x[no_instances:]

    start_time = time.time()
    if method == 'ols':
        reg = LinearRegression()
    elif method == 'lasso':
        reg = Lasso(alpha=0.1)
    reg.fit(x_train, y_train)
    score = reg.score(x_test, y_test)
    elapsed_time = time.time() - start_time

    return dict(
        score = score,
        elapsed_time = elapsed_time,
    )

do_simulation_study(to_sample, func, db, Result,
max_count=no_simulations)
```

Source code 3 – Part of the simulation study results exploration file

```
import pandas as pd
...
df = pd.DataFrame(list(Result.select().dicts()))
df.groupby(['data_distribution', 'no_instances',
'method']).mean()
```

3.3.4 Main features and documented examples

In the package documentation available at <https://sstudy.marcoinacio.com/>, we present the following features and examples:

- Support to SQLite, PostgreSQL, MySQL and CockroachDB (and, at least in principle, any additional dataset supported by *peewee*).
- Automatic randomization of executions.
- Optional filter of undesired simulation options.
- Prevention of SQL server disconnect failures: waits for availability of the server again so that long simulation calculations are not lost.
- Automatic handling of binary data: whenever a dataset field is a *BlobField*, invokes the "binarizer" *pickle.dumps* automatically. This allows the user to store whole arrays or large class instances as results into the SQL database.
- Hints on exploring the results using *pandas* package.

3.4 Examples of applications

In the following subsections, we present a series of the example of usage of simulation studies using the *sstudy* package, the source code of all example is available to download in the package *examples* folder at <https://gitlab.com/marcoinacio/sstudy/-/tree/master/examples>. A Dockerfile is also available at <https://gitlab.com/marcoinacio/sstudy/-/blob/master/Dockerfile> in order to install the dependencies and run all examples on Docker.

3.4.1 Simple regression

Suppose that we want to compare the performance of ordinary least squares with the performance of a LASSO with data being generated from a Gaussian linear regression: e.g.: each dataset contains 100 instances $(X_1, X_2, ..., X_{100})$, with each instance arising independently from a $Y|X \sim \text{Gaussian}(X\beta, \sigma)$. $X \sim \text{Multivariate Gaussian}(0, 2I)$.

In other to proceed with the evaluation, one must note first that there are many possibilities that we could setup here in order to test the estimators performance: we could compare the estimated values of $\mu = X\beta$ or compare directly the estimated y's. Moreover, we can also choose from a wide range of loss criteria[3] like the mean squared error, mean absolute error, etc.

A second point to notice here is that one might be tempted to generate a single train dataset D_{train} (i.e.: a sample $(X_1, X_2, ..., X_{30})$) and a single test dataset D_{test} (i.e.: a sample $(X_1, X_2, ..., X_m)$ as large as we want), fit the models (empirical mean and median) to data, and evaluate their mean squared error. In this case however, we would be affected by random chance and would be unable to conclude with certainty which model bests adjust to the data: maybe model A is better for this true distribution, but by chance it happened to obtain a bad fit this specific dataset that was generated.

If however, we try to solve this problem by means of increasing the train dataset size, then we fall into another problem: we would be concluding which model better fits a large sample size instead of perceiving their behaviour on smaller samples[4], but in the real world, we do not have access to an infinite amount of data.

The solution given by a simulation study is to repeat such a procedure many times to the point that in the long run, we are able to distinguish which model is the best for this true distribution and this decision criteria even in the presence of small datasets.

In Table 1, we present the results for such a simulation experiment (in parenthesis we present the standard error of measurement of the simulation study, note that if you increase the number of simulations, the standard error will tend to zero by the law of large numbers).

The source code for this experiment is available to download in the package *examples/basic* folder at <https://gitlab.com/marcoinacio/sstudy/-/tree/master/examples/basic>.

3.4.2 P-values of hypothesis tests

Simulation studies can also be used to compare the hypothesis testing methods (see Inácio, Izbicki and Gyires-Tóth (2020) and Inácio, Izbicki and Stern (2019), for instance). In this case, two important criteria arises: the uniformity of the test under the null hypothesis and the test power under the alternative hypothesis.

Given a dataset D_{train} (i.e.: a sample $(X_1, X_2, ..., X_n)$) with each instance coming independently from a Gaussian$(\mu = 2, 1)$, we could, for instance, compare the tests type

[3] Without loss of generality, you can also work with utility, score and other decision criteria.
[4] Note that models that have terrible behaviour on small datasets, might get increasingly better as the sample size increases (i.e.: bad estimators might be consistent). An example would be the estimator $\sum_{i=1}^{n} x_i)/(n - 10000)$ which is generally bad for small samples but equals to the empirical mean as n approaches infinity.

Table 1 – Results for a simulation experiment using a Gaussian linear regression as dataset generator using MSE on holdout set as score.

data distribution	n. of instances	method	score
complete	100	LASSO	0.803 (0.035)
		ols	0.838 (0.008)
	1000	LASSO	0.856 (0.011)
		ols	0.834 (0.010)
	10000	LASSO	0.842 (0.016)
		ols	0.825 (0.012)
sparse	100	LASSO	0.608 (0.073)
		ols	0.660 (0.022)
	1000	LASSO	0.747 (0.049)
		ols	0.702 (0.022)
	10000	LASSO	0.688 (0.040)
		ols	0.695 (0.021)

1 error rate of method A and B under the null hypothesis $\mu = 2$ and compare the test power of such methods under the alternative hypothesis $\mu = 3.5$.

Additionally, we could change the true distribution to something other than a Gaussian to verify how that affects the type I error and the test power.

In order to illustrate this, we present an simulation study comparing the performance of two sample comparison methods, i.e.: hypothesis tests that take two datasets as input and attempt to test the hypothesis of whether the two datasets arise from the same data generating function. We work with methods Mann-Whitney rank test(MANN; WHITNEY, 1947), Kolmogorov-Smirnov(SMIRNOV, 1948) and Welch's t-test(WELCH, 1947) for datasets with 1000 and 2000 instances, with each instance being generated by a standard log-normal distribution. Moreover, under the alternative hypothesis, one the datasets has 0.1 added to all of instances after sampling from the log-normal distribution. Beyond that, 1000 simulations were performed for each configuration.

In Figure 5, we present the empirical cumulative distribution of the p-values under the null hypothesis while in Figure 6, we present empirical cumulative distribution of the p-values under the alternative hypothesis, which can also be interpreted as the test power. We also present the confidence bandwidth of two times the standard error (approximately 95% asymptotically). We also present the results in Tables 2 and 3.

As can be seemed from the results, all tests are well behaved in terms of being uniform under the null hypothesis as well as having test power that increases with number of instances, with the Kolmogorov-Smirnov test outperforming both tests and the Mann-Whitney test outperforming the Welch's t-test.

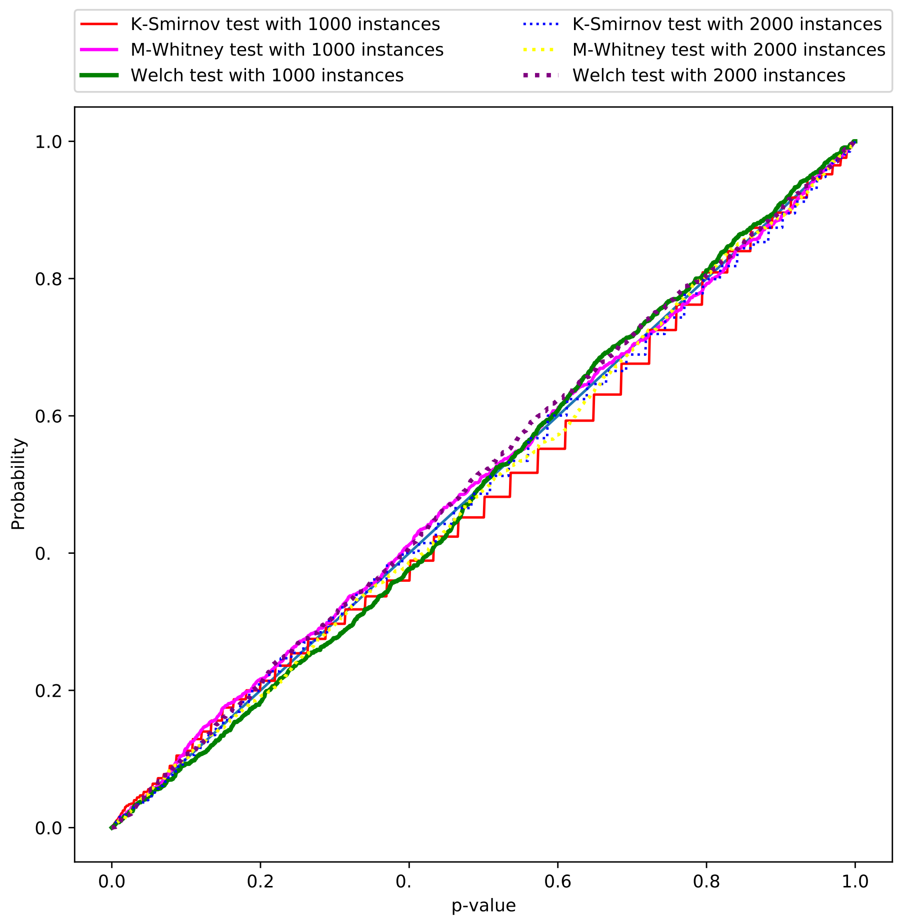


Figure 5 – Null hypothesis p-value distribution.

Table 2 – Test type I error rates.

method	n instances	avg p-value	Error (α=1%)	Error (α=5%)
K-Smirnov	1000	0.511 (0.009)	0.016 (0.004)	0.056 (0.007)
K-Smirnov	2000	0.505 (0.009)	0.015 (0.004)	0.044 (0.006)
M-Whitney	1000	0.494 (0.009)	0.012 (0.003)	0.049 (0.007)
M-Whitney	2000	0.506 (0.009)	0.010 (0.003)	0.050 (0.007)
Welch	1000	0.502 (0.009)	0.008 (0.003)	0.047 (0.007)
Welch	2000	0.490 (0.009)	0.006 (0.002)	0.055 (0.007)

Table 3 – Test power.

method	n instances	avg p-value	Power (α=1%)	Power (α=5%)
K-Smirnov	1000	0.019 (0.001)	0.651 (0.015)	0.887 (0.010)
K-Smirnov	2000	0.001 (0.000)	0.991 (0.003)	1.000 (0.000)
M-Whitney	1000	0.052 (0.004)	0.580 (0.016)	0.791 (0.013)
M-Whitney	2000	0.004 (0.001)	0.910 (0.009)	0.980 (0.004)
Welch	1000	0.348 (0.009)	0.062 (0.008)	0.190 (0.012)
Welch	2000	0.256 (0.009)	0.127 (0.011)	0.333 (0.015)

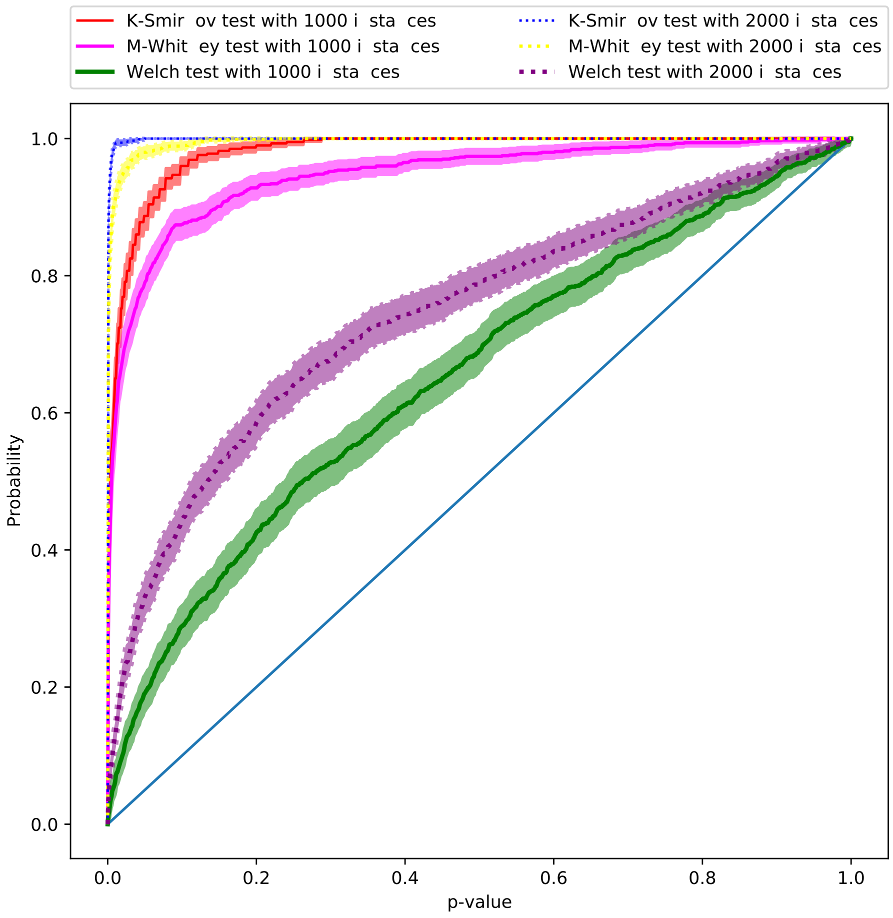


Figure 6 – Test power.

The source code for this experiment is available to download in the package *examples/hypothesis_testing* folder at <https://gitlab.com/marcoinacio/sstudy/-/tree/master/examples/hypothesis_testing>.

3.4.3 Neural networks and non-deterministic estimators

For neural networks and other non-deterministic estimators, in general, we also randomize the initialization parameters of the estimator[5]

In this case, suppose that the method becomes deterministic given a vector of parameters[6] β, we would then be estimating:

$$E_{D \in \mathbb{D}_P} \left[\text{loss} \left(M_{\{k,\beta\}}, D \right) \right]$$

Therefore, the non-determinism of the method would be "averaged out" after a large number of simulations and the same conclusions would follow as was previously done.

[5] e.g.: for neural networks, using random Xavier (GLOROT; BENGIO, 2010b) or Kaimining (HE *et al.*, 2015) initializations.

[6] For neural networks, that would be the initial value of its neurons.

In order to illustrate this, we present an simulation study comparing the performance of neural networks given distinct number of hidden layers, with or without dropout (SRIVASTAVA *et al.*, 2014) and on training and test datasets.

We work with the following true distribution:

$$\begin{aligned}
X_{i,1} &\sim \text{Gaussian}(0,1)\\
X_{i,2} &\sim \text{Gaussian}(0,1)\\
Y_i &\sim \cos(X_{i,1}) + \sin(X_{i,2}) + e_i\\
e_i &\sim \text{Gaussian}(0,1)
\end{aligned}$$

With each train dataset composed of $((X_1,Y_1),(X_2,Y_2),...,(X_{1000},Y_{1000}))$ and each test dataset composed of $((X_{1001},Y_{1001},(X_{1002},Y_{1002}),...,(X_{2000},Y_{2000}))$, both with instances sampled i.i.d. We work with standard dense neural networks with 2 hidden layers of the same size, ELU activations (DJORK-ARNÉ; UNTERTHINER; HOCHREITER, 2016) and batch normalization (IOFFE; SZEGEDY, 2015), moreover we use Pytorch (PASZKE *et al.*, 2019) as neural networks framework with *nnlocallinear* Python package (COSCRATO *et al.*, 2019b) on top it.

In Figures 7, 8, 9 and 10 we present the results of the experiment when with and without dropout and/or batch normalization. When using dropout, we set a 0.5 dropout rate.

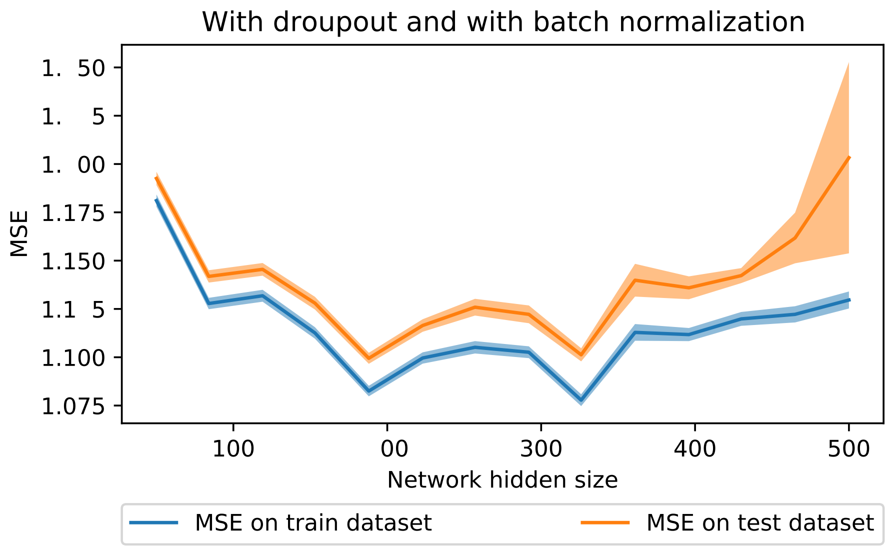


Figure 7 – MSE for distinct hidden sizes, with dropout and with batch normalization

The source code for this experiment is available to download in the package *examples/neural_networks* folder at <https://gitlab.com/marcoinacio/sstudy/-/tree/master/examples/neural_networks>.

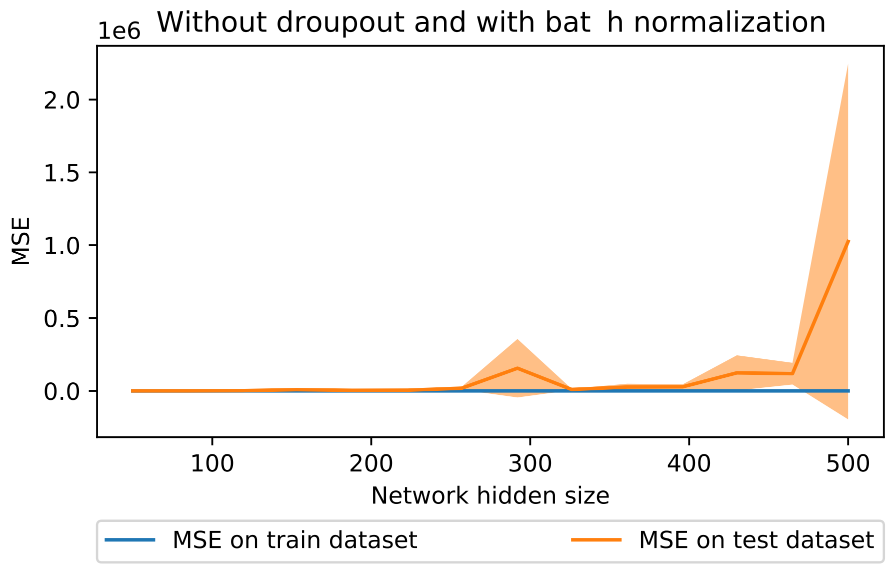


Figure 8 – MSE for distinct hidden sizes, without dropout and with batch normalization

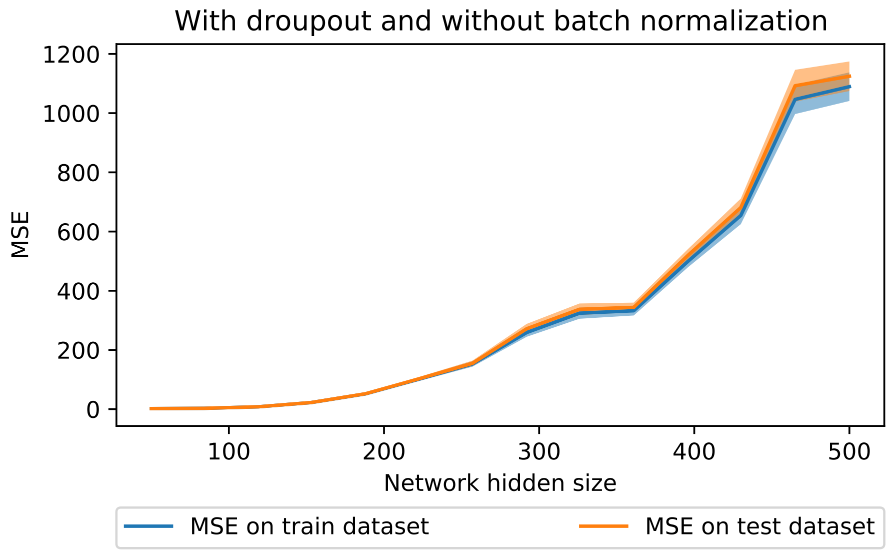


Figure 9 – MSE for distinct hidden sizes, with dropout and without batch normalization

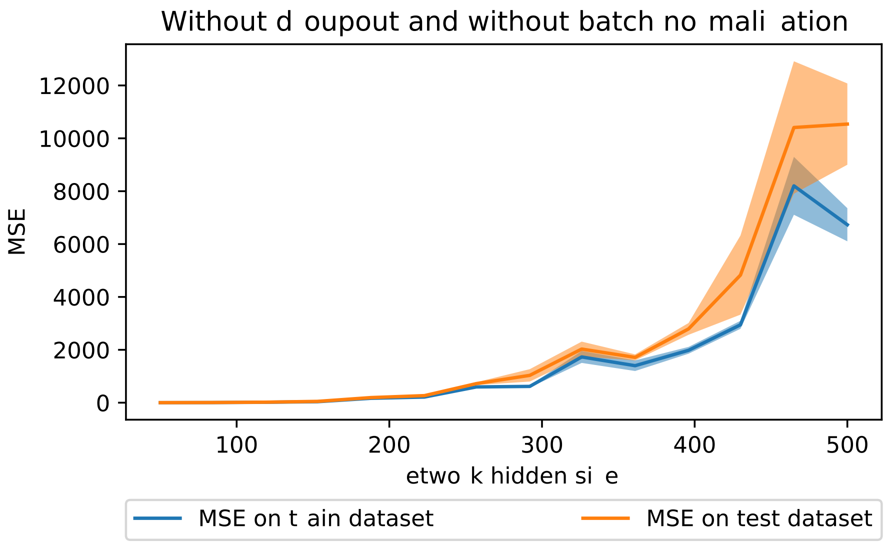


Figure 10 – MSE for distinct hidden sizes, without dropout and without batch normalization

3.4.4 Density estimation and Bayesian inference

Another interesting instance for using simulation studies is on density estimation. Moreover, Bayesian estimators in general can also have their frequentist properties evaluated using simulation studies (RUBIN, 1984).

To illustrate both points, we compare the performance of a npcompare(INÁCIO; IZBICKI; SALASAR, 2018): a Bayesian density estimator against the kernel density estimator (ROSENBLATT, 1956; PARZEN, 1962; SILVERMAN, 1986) with bandwidth hyper parameter chosen by data splitting.

We work with a mixture of gammas as true distribution for generating the dataset $(Y_1, Y_2, ..., Y_n)$:

$$
\begin{aligned}
X_{i,1} &\sim \text{Beta}(1.3, 1.3) \\
X_{i,2} &\sim \text{Beta}(1.1, 3.0) \\
X_{i,3} &\sim \text{Beta}(5.0, 1.0) \\
X_{i,4} &\sim \text{Beta}(1.5, 4.0) \\
P(Y_i = X_{i,1}) &= 0.2 \\
P(Y_i = X_{i,2}) &= 0.25 \\
P(Y_i = X_{i,3}) &= 0.35 \\
P(Y_i = X_{i,4}) &= 0.2
\end{aligned}
$$

Moreover, we use the integrated squared loss as loss function:

$$\int_0^1 (f(x) - \hat{f}(x))^2 \mathrm{d}x$$

Note that in this case, there is no test dataset and the loss is evaluated directly against the true distribution.

In Table 4, we present the results of the experiment

Table 4 – Results for a density estimation experiment.

number no instances	method	loss	number simul
100	kde	0.107 (0.003)	500
	npcompare	0.038 (0.005)	30
200	kde	0.068 (0.002)	500
	npcompare	0.021 (0.002)	30

As can be seemed, the *npcompare* method outperfomed the *kde* for with both 100 and 200 instances. Note that we used a lower number of simulations for the *npcompare* method to its higher computational time, however, this was enough to notice the superiority of the method (for this true distribution) given the calculated standard error.

The source code for this experiment is available to download in the package *examples/density_estimation* folder at <https://gitlab.com/marcoinacio/sstudy/-/tree/master/examples/density_estimation>.

3.5 Conclusion

In this chapter, we have presented a Python package called *sstudy*, designed to simplify the preparation of simulation studies; we presented its basic features, usage examples and references to the its documentation. Moreover, we also presented a short statistical description of the simulation study procedure as well as usage examples.

CHAPTER

4

DISTANCE ASSESSMENT AND ANALYSIS OF HIGH-DIMENSIONAL SAMPLES USING VARIATIONAL AUTOENCODERS

An important question in many applications of machine learning and Statistics is whether two samples (or datasets) arise from the same data generating probability distribution (GRETTON A.AND BORGWARDT *et al.*, 2012; HOLMES *et al.*, 2015; SORIANO, 2015). Although an old topic in statistics (MANN; WHITNEY, 1947; SMIRNOV, 1948), simple accept/reject decisions given by most hypothesis tests are often not enough: it is well known that the rejection of the null hypothesis does not mean that the difference between the two groups is meaningful from a practical perspective (COSCRATO *et al.*, 2019a; WASSERSTEIN; SCHIRM; LAZAR, 2019). Thus, tests that go beyond accept/reject decisions are preferred in practice. In particular, tests that provide not only single and interpretable numerical values, but also a visual way of exploring how far apart the datasets are from each other especially useful. This raises the question of how to assess the distance between two groups meaningfully, which is especially challenging in high-dimensional spaces.

In this chapter, we present a novel nonparametric approach to assess the dissimilarity between two high-dimensional datasets using variational autoencoders (VAE) (KINGMA; WELLING, 2014). We show how our approach can be used to visually assess how far apart datasets are from each other via a boxplot of their distances and additionally, provide a way of interpreting the scale of these distances by using the distance between known distributions as a baseline. We also show how a formal permutation-based hypothesis testing can be derived within our framework.

The remaining of this chapter is organized as follows. In sections 4.0.1 and 4.0.2, we present a brief description of articles related to our proposed method. In Section 4.1, we present a review of variational inference and VAE, and show that the latter can be interpreted as a density estimation procedure, which is the basis of the proposed method. In Section 4.2, we show how variational autoencoders can be used as a method of exploring the differences between two samples. In Section 4.3, we use our method to derive a formal hypothesis testing procedure. Both sections also show applications of the methods to simulated and real-world datasets. Finally, Section 4.4 concludes the paper with final remarks. Appendix 4.4 contains details on the configurations of the software and neural networks used, as well as a link to our implementation, which is published open source.

4.0.1 Related work on two-sample hypothesis testing

Several nonparametric two-sample testing methods have been proposed in the literature; they date back to Mann and Whitney (1947), Smirnov (1948), Welch (1947): three classical two-sample tests (Mann-Whitney rank test, Kolmogorov-Smirnov and Welch's t-test, respectively) which were designed to work for univariate random variables only. On the other hand, Holmes *et al.* (2015), Soriano (2015), Ceregatti, Izbicki and Salasar (2018) investigate Bayesian univariate methods for this task.

More recently, Gretton A.and Borgwardt *et al.* (2012) introduce a two-sample test comparison using reproducing kernel Hilbert space theory that works for high-dimensional data. The test, however, does not provide a way to visually assess the dissimilarity between the datasets. Kirchler *et al.* (2020) proposes a method for two-sample hypothesis testing utilizing deep learning, which contrary to a permutation based test, only controls the type-1 error rate asymptotically; Lopez-Paz and Oquab (2017) proposes a test statistic built using binary classifier in the context of causal inference and causal discovery, also relaying on asymptotic distribution for the test statistic (the distance between the performance of binary classifiers) under the null hypothesis.

Other two-sample tests for high-dimensional data can be found in (MONDAL; BISWAS; GHOSH, 2015; ZHOU *et al.*, 2016) and references therein. Although these tests are robust and effective in many settings, they do not provide a visual analysis to assess the distance between the groups. Thus, they do not provide ways of checking if the difference between the datasets is meaningful from a practical perspective, a gap in literature which is filled by this chapter.

4.0.2 Related work on two-sample comparison and distance measurement

There has also been some work devoted to two-sample comparison and related tasks: In particular Inácio, Izbicki and Salasar (2018), provides a framework for assessing the distance between populations using density estimation methods. However, the method provided in that work is based on MCMC (Markov Chain Monte Carlo) Bayesian simulations, and therefore it is unable to scale to large datasets (see Betancourt (2015), for instance) and high-dimensional spaces. In this chapter, we overcome these issues by using variational autoencoders to estimate densities, and by introducing a specific metric which has an analytic solution even in high-dimensional spaces.

Kornblith *et al.* (2019) (and references therein) proposes a new method of comparison of neural networks representation. Larsen *et al.* (2016) propose a variant of variational autoencoders (VAE) that better measure similarities in data space than a vanilla VAE. An and Cho (2015) uses VAEs for anomaly detection: that is, with the goal of identifying whether a single instance is different from an observed sample. Dengsheng Zhang and Guojun Lu (2003) evaluates existing similarity measurement methods in the context of image retrieval.

These papers however do not use their methods for performing formal hypothesis tests.

Finally, for closely-related problems and applications, see also Pfister *et al.* (2016), Ramdas, Trillos and Cuturi (2017), Inácio, Izbicki and Stern (2019) for methods on how to solve the problem of independence testing and Bińkowski *et al.* (2018) which uses two-sample tests as a tool to evaluate generative adversarial networks.

4.1 Variational Autoencoders

In this section, we review key aspects of the variational autoencoders framework (KINGMA; WELLING, 2014) which are important to our proposed method.

Variational autoencoders are among the most famous deep neural network architectures. The generative behaviour of VAEs makes these model attractive for many application scenarios. VAEs are often used in computer vision related tasks. Introducing labeled data to the VAE training, attribute vectors, such as smile vector (YAN *et al.*, 2016), can be computed; i.e. in Yan *et al.* (2016) the smile vector is computed by subtracting the mean latent vector for images of smiling and non-smiling people. In the generation phase, this vector can be altered in the latent space to generate faces with different smiling attributes. Another work utilizes VAEs to predict the possible movement of objects on images, pixelwise (WALKER *et al.*, 2016). Videos were also generated from text by combining VAEs with Generative Adversarial Network (GANS) (LI *et al.*, 2018). VAEs are also successfully applied in speech technologies. Hsu, Zhang and Glass (2017) learns latent representations from unlabelled data with VAEs for speech transformation (including phonetic content and speaker identity). In text-to-speech synthesis systems VAEs can be successfully applied for learning attributes and thus, controllable, expressive speech can be generated (AKUZAWA; IWASAWA; MATSUO, 2018). Other types of sequences, like text, can also be modeled with VAEs. Semeniuta, Severyn and Barth (2017) uses convolutional encoder and deconvolutional decoder components, augmented with a recurrent language model in a variational autoencoder architecture to model text. Furthermore, there have been numerous theoretical research that focuses on or utilizes VAEs, like for second-order gradient estimation (FAN *et al.*, 2015), for importance weighting (BURDA; GROSSE; SALAKHUTDINOV, 2015), for anomaly detection (SUH *et al.*, 2016) and for novel architectures, like ladder VAE (SØNDERBY *et al.*, 2016).

4.1.1 Statistical definition

Consider an i.i.d. random sample $D = (X_1, X_2, ..., X_n)$. Variational autoencoders estimate the density of this sample by encoding the information of each X_i using latent random variables $Z = (Z_1, Z_2, ..., Z_n)$, which are linked to $(X_1, X_2, ..., X_n)$ by a parameter θ. More precisely, the model assumes the structure

$$P_\theta(D = d|Z) = \prod_{i=1}^{n} \mathscr{N}(X_i = x_i; (\mu_i, \sigma_i) = g_\theta(Z_i)),$$

where $Z_i \sim \mathscr{N}(0,1)$, g_θ is a complex function (a neural network) with parameter θ (i.e.: the parameters/weights of a neural network), and μ_i and σ_i are the mean and standard deviation of the Gaussian distribution. Inference on such model is performed by maximizing the evidence $P(D = d; \theta) := P_\theta(D = d)$.

Note that, if g_θ is complex enough, we can actually model any distribution of X_i (DEVROYE, 1986). This is why g_θ is parametrized using an artificial neural network; it leads to flexibility (because of the richness of the space of functions they can represent, Hornik, Stinchcombe and White (1989)) as well as scalability.

Unfortunately, maximization of the evidence cannot be directly solved due to the curse of dimensionality (DOERSCH, 2016). The next section shows how variational inference can be used to overcome this.

4.1.2 Variational inference

The curse of dimensionality can be solved using variational inference, which consists of optimizing

$$\begin{aligned} &\log P_\theta(D = d) - \mathbf{D}_{KL}(Q_\phi^{(Z|D=d)} | P_\theta^{(Z|D=d)}) \\ &= E_{Q_\phi}[\log P_\theta(D|Z) | D = d] \\ &- \mathbf{D}_{KL}(Q_\phi^{(Z|D=d)} | P^{(Z)}) \end{aligned}$$

where $\mathbf{D}_{KL}$ refers to the Kullback-Leibler divergence and Q given by:

$$Q_\phi(Z_i|X_i = x_i) = \mathscr{N}(Z_i; (\mu_i, \sigma_i) = h_\phi(x_i))$$

This framework is called variational autoenconder and it solves the curse of dimensionality (KINGMA; WELLING, 2014). The training procedure for variational autoencoders is presented in Figure 11; for more details, see Kingma and Welling (2014).

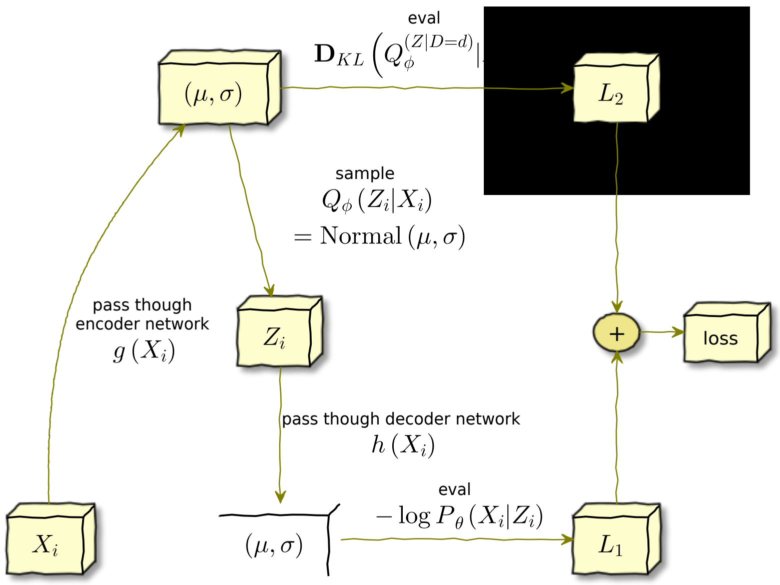


Figure 11 – VAE training procedure.

4.1.3 Generative model

The trained model can be used to generate new instances $\widetilde{X}_j$: this can be done by applying $P_{\hat{\theta}}(\widetilde{X}_j) = \int P_{\hat{\theta}}(\widetilde{X}_j|\widetilde{Z}_j = z)P^{(\widetilde{Z}_j)}(\mathrm{d}z)$, i.e.: sample $Z \sim N(0,1)$, apply it on the neural network g_θ and then sample from a $\mathcal{N}((\mu,\sigma) = g_\theta(z))$. Therefore, variational autoencoders are a Gaussian[1] mixture model (of an infinite number of Gaussians), and thus a density estimator.

4.1.4 Identifiability of the mixture of Gaussians

As per the structure of variational autoencoders, the distribution of such mixture of Gausssians is not identifiable (TEICHER *et al.*, 1961; WECHSLER; IZBICKI; ESTEVES, 2013): two different configurations of the parameters, say θ_1 and θ_2, can lead to the exact same distribution of (μ,σ). That is, $g_{\theta_1}(Z) \sim g_{\theta_2}(Z)$ even if $\theta_1 \neq \theta_2$.

In other words: if we train a variational autoencoders framework on a dataset, we will get a "generator" of pairs (μ,σ), say m_1; and if we train a variational autoencoders framework with identical structure on the same dataset, we will get another "generator" of pairs (μ,σ), say m_2. Generators m_1 and m_2 do not necessarily give the same distribution

[1] Note that variational autoencoders setup can be used with distributions other then Gaussians; e.g.: discrete data with Bernoulli distribution.

over samples of pairs (μ, σ). Nonetheless, the induced final density (i.e., the Gaussian mixture) should be same analytically (i.e.: ignoring the stochastic variation that estimation methods induce).

4.2 Two sample comparison: definition of the distance

We name our approach for assessing the similarity between two datasets, D_1 and D_2 as *vaecompare* and describe it as follows. First, we train two variational autoencoders: one for D_1 and one for D_2. Let g_{θ_1} and g_{θ_2} be the learned functions for each of the autoencoders. g_{θ_1} and g_{θ_2}, together with $Z \sim N(0,1)$, induce two distributions over the parameter space (μ, σ). Let $S_1 = (\mu_1, \sigma_1)$ and $S_2 = (\mu_2, \sigma_2)$ be two samples generated from the enconders g_{θ_1} and g_{θ_2}, respectively. We then measure the distance between S_1 and S_2. Now, recall that each (μ, σ) is used to generate a new sample $X \sim N(\mu, \sigma)$ (Section 4.1.3). Thus, a meaningful distance between S_1 and S_2 should be in the space of the random variables they generate. The key idea to make the method computationally feasible is to use a symmetric Kullback-Leibler divergence between the distributions induced by S_1 and S_2:

$$\mathbb{D}(S_1, S_2) := \frac{\mathbf{D}_{KL}(P_{S_1}, P_{S_2}) + \mathbf{D}_{KL}(P_{S_2}, P_{S_1})}{2d},$$

where d is the dimension of the feature space, P_{S_i} is a (multivariate) Gaussian distribution with parameters (μ_i, σ_i), and $\mathbf{D}_{KL}$ is the Kullback-Leibler divergence. $\mathbf{D}_{KL}$ has an analytical solution in the Gaussian case:

$$\mathbf{D}_{KL}(\mathcal{N}(\mu_1, \sigma_1^T I), \mathcal{N}(\mu_2, \sigma_2^T I)) =$$
$$\frac{1}{2}\left[2\left(\sum_{i=1}^{d} \log \sigma_{2,i} - \log \sigma_{1,i}\right) - d + \left(\sum_{i=1}^{d} \sigma_{1,i}^2 / \sigma_{2,i}^2\right) + \left(\sum_{i=1}^{d} \sigma_{2,i}^2 (\mu_{2,i} - \mu_{1,i})^2\right)\right]$$

In case X represents an image, we use the standard approach of using multi-dimensional Bernoulli distributions with dimensions independent from each other (see Kingma and Welling (2014), Doersch (2016), for instance). In this case, the Kullback-Leibler can also be obtained analytically:

$$\frac{1}{2d}\left(\mathbf{D}_{KL}(\text{Bernoulli}(p), \text{Bernoulli}(q)) + \mathbf{D}_{KL}(\text{Bernoulli}(q), \text{Bernoulli}(p))\right)$$
$$= \frac{1}{2d}\sum_{i=1}^{d}(q_i - p_i)(\log(q_i) - \log(p_i) + \log(1-p_i) - \log(1-q_i))$$

Using this approach, we can therefore assess the distance between a sample generated from the first autoencoder and a sample generated from the second autoencoder. In order to assess the divergence between the datasets D_1 and D_2, we can repeat this procedure several times; this will give a sample of the distribution of distances.

Now, in order to overcome the identifiability issue discussed in Section 4.1.4, we train the variational autoencoders multiple times (we call these "refits") for each dataset (using distinct initialization seeds for the network parameters) and use the new instances pairs (μ, σ) from each of them in equal proportion. The full procedure is summarized in Algorithm 2 and Figure 12.

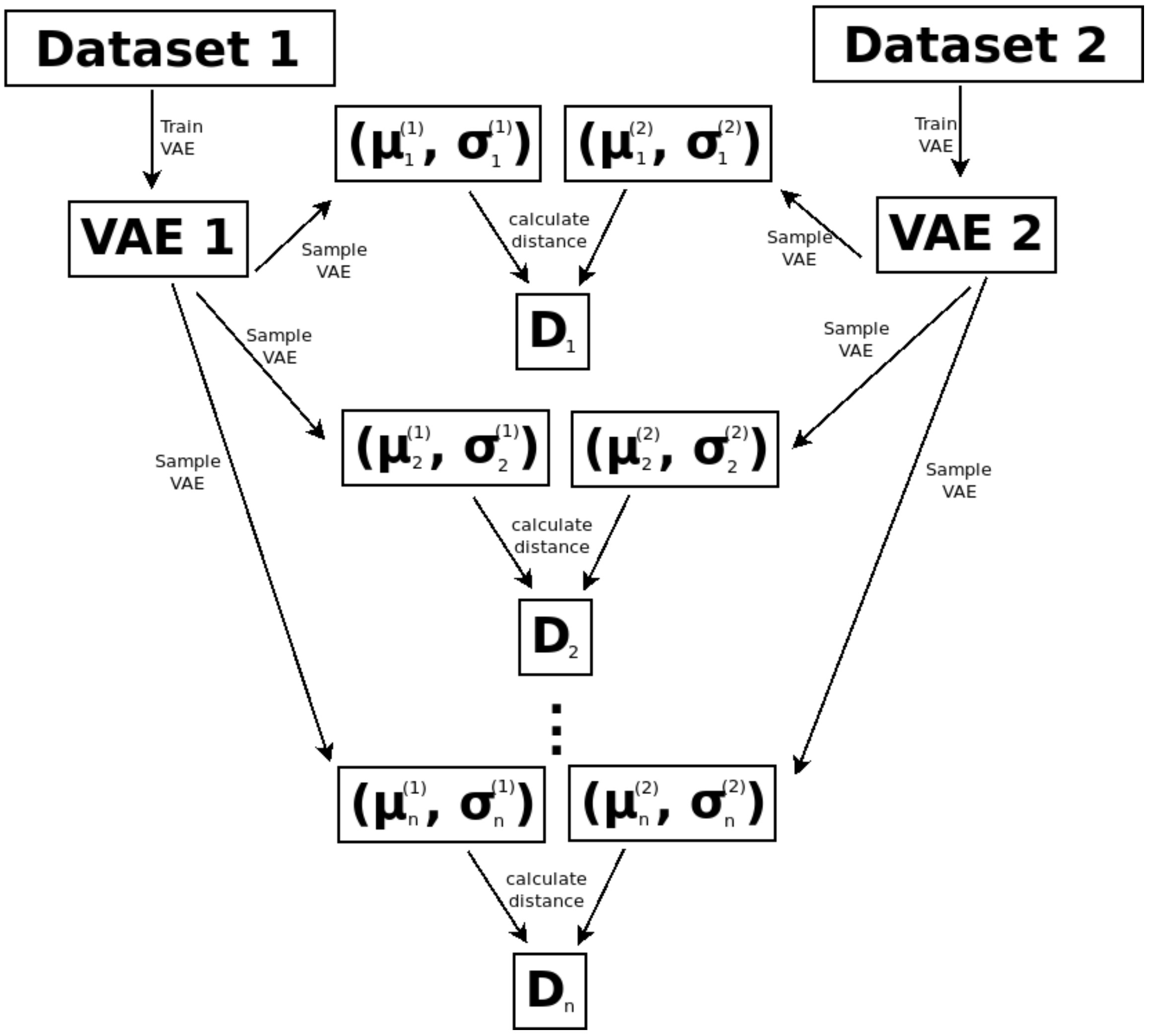

Dataset 1
Dataset 2
Train VAE
Train VAE
VAE 1
VAE 2
Sample VAE
Sample VAE
Sample VAE
Sample VAE
Sample VAE
Sample VAE
$(\mu_1^{(1)}, \sigma_1^{(1)})$
$(\mu_1^{(2)}, \sigma_1^{(2)})$
calculate distance
D_1
$(\mu_2^{(1)}, \sigma_2^{(1)})$
$(\mu_2^{(2)}, \sigma_2^{(2)})$
calculate distance
D_2
$(\mu_n^{(1)}, \sigma_n^{(1)})$
$(\mu_n^{(2)}, \sigma_n^{(2)})$
calculate distance
D_n

Figure 12 – Schematic representation of the procedure to generate divergence samples for data comparison.

Note that from the perspective of applying this method to images, it can also be interpreted as a data exploration tool, as it helps exploring the separability and uncertainty of classes of images and the relation between their data generating processes.

4.2.1 Assessing the magnitude of the distance

In Section 4.2, we defined a method to measure the distance between two datasets. A yardstick is still required in order to say what is a "low" and "high" distance. In order to create a baseline to interpret such distances, we proceed in similar fashion as Inácio, Izbicki and Salasar (2018): we compute the distance between two known distributions.

Algorithm 2 – Generating divergence samples using *vaecompare*

Input: dataset D_1, dataset D_2, number of desired samples per refit n, number of desired refits R

Output: divergence samples S.

```
1: for $i \in \{1,\dots,R\}$ do
2:     Train VAE $V_1$ from $D_1$
3:     Train VAE $V_2$ from $D_2$
4:     for $j \in \{1,\dots,n\}$ do
5:         Generate a sample $s_1$ from $V_1$ (e.g.: a pair $(\mu,\sigma)$ for Gaussian VAE).
6:         Generate a sample $s_2$ from $V_2$.
7:         Calculate $\mathbb{D}(s_1,s_2)$ and store it on $S$.
8:     end for
9: end for
```

In the case of Gaussian VAEs we can work for instance with $\mathbb{D}(\mathcal{N}_0,\mathcal{N}_1)$, where $\mathcal{N}_0$ is a multivariate Gaussian with covariance given by an identity matrix and mean given by a vector of zeros and $\mathcal{N}_1$ is a multivariate Gaussian with covariance given by an identity matrix and mean given by a vector of ones. We have that $\mathbb{D}(\mathcal{N}_0,\mathcal{N}_1) = 1/2$. For binomial VAEs, we use known binomial distributions as the baseline.

4.2.2 Evaluation (images)

Next, we apply the method to CIFAR10 data (KRIZHEVSKY, 2009) using the VAE as a generator of binomial distributions. The dataset consists of images from 10 distinct categories (ranging from 0 to 9), with each category containing 5000 images. To make the comparison fair when comparing a category to itself and when comparing a category to another, we chose to work with half of each category dataset (2500 images) to train each VAE; i.e.: when comparing category 0 to category 1, one VAE is trained with 2500 images from category 0 and the other is trained with 2500 images from category 1; on the other hand, when comparing category 0 to itself, each VAE is trained with half (2500 images) of the category 0 dataset. We worked with 90 VAE refits for each dataset.

In Figure 13, we present the results of such experiment with boxplots of the obtained divergences for all possible category combinations (note that the lower image of each plot is a zoomed-in version of the upper image). The figure also shows the median and mean for each category, as well as the divergence of known Bernoulli distributions (plotted as horizontal lines).

Except for categories 0, 2 and 4, the divergence samples were all concentrated near zero when comparing a category to itself (a desirable behaviour). On the other hand, for these 3 categories, we can observe a considerable amount of divergence samples spread far from zero indicating some uncertainty, but even in this case, there was a considerable amount of them near zero. Note also that the median for these three categories is much

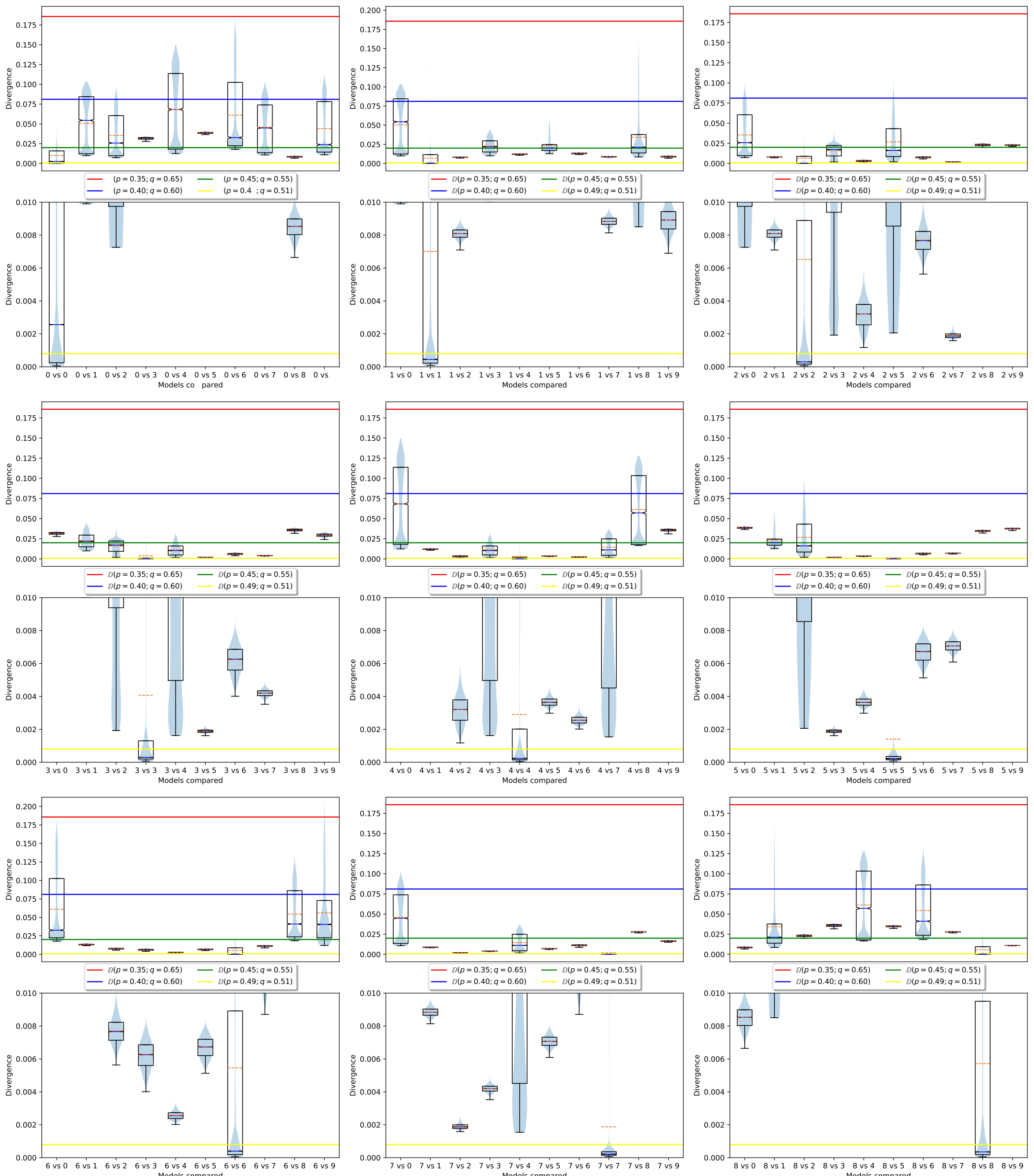


Figure 13 – Box plots of samples from our divergences comparing categories 0 to 8 to all categories. Note that the lower image of each plot is a zoomed-in version of the upper image.

closer to zero than the mean, indicating its resilience to outliers.

On the other hand, when making comparisons between distinct categories, there are cases with high uncertainty (i.e.: boxplots with wide extensions; generally with few points close to zero), as well as cases with higher certainty (i.e.: boxplots with narrow extensions).

We conclude that the method is therefore useful for the purpose of data exploration as it works as expected in a complex space such as images.

4.3 Hypothesis testing

We can additionally use *vaecompare* to directly test if two samples come from the same population. One way to do this is to find a threshold value (*cutpoint*) from a decision theoretic stand point where we would reject the null hypothesis of the two samples coming from the same population. This is what is done in (CEREGATTI; IZBICKI; SALASAR, 2018) in the case of a Dirichlet process prior, where the threshold is chosen so as to control type I error of the hypothesis test. Unfortunately, this is not possible in general and in general the cutpoint depends on the true data generating function.

Given that, we work instead with a simple permutation test where the datasets are repeatedly permuted against each other (i.e.: their data is mixed), and the average divergence of the samples is used as a test statistic. The p-value is then given by the quantile of the non-permuted dataset among all the statistics[2]. We note that for the hypothesis test to work in the sense of being a proper test (uniform under the null), it is not necessary to do VAE refits, but refits increase the test power as we shall see next. We present the procedure in Algorithm 3.

4.3.1 Evaluation (simulated data)

In this section, we apply the proposed hypothesis testing method to simulated datasets from a known data generating function and plot the observed p-value distribution. The data generating function for the datasets is defined as:

$$\begin{aligned}&\mathrm{lgr}(\mu = log(2), \sigma = \alpha) - \mathrm{lgr}(\mu = log(2), \sigma = 0.5)\\&+ \mathrm{gr}(\mu = 1, \sigma = 2) + k\end{aligned}$$

where lgr stands for multivariate log Gaussian random number generator, and gr stands for multivariate Gaussian random number generator, both with diagonal covariance matrices. Moreover, $\alpha = \{0.2 + 0.7 * i/9\}_{i=0}^{i=9}$, i.e.: $\alpha_i = 0.2 + 0.7 * i/9$ for $i \in \{0, 1, ..., 9\}$.

[2] For instance, if 43 of the permuted datasets had resulted on a lower divergence statistic than that of non-permuted dataset and on the other hand 57 had resulted on greater divergence, then the p-value would be $43/(43+57) = 0.43$.

Algorithm 3 – Obtaining the p-value for hypothesis testing using *vaecompare*

Input: dataset D_1, dataset D_2, number of desired samples per refit n, number of desired refits R, number of permutations t, averaging function M (e.g. mean or median)

Output: p-value ρ.

```
1: for $i \in \{1,\dots,t\}$ do
2:     Run Algorithm 2, and store the results in $S_i$.
3:     Calculate $M(S_i)$ and store the result in $K_i$.
4:     Permute the instances of datasets $D_1$ and $D_2$.
5: end for
6: Obtain the number of points $q_1$ in $\{K_2, K_3, ..., K_t\}$ which are greater than $K_1$.
7: Obtain the number of points $q_2$ in $\{K_2, K_3, ..., K_t\}$ which are greater than or equal to
   $K_1$.
8: Set $q = (q_1 + q_2)/2$
9: Store $(q+1)/(t+1)$ in $\rho$.
```

For simplicity, we do not use refits here. The value of the vector k is fixed in zero for one of the datasets, and varied for the other. This is done in order to change the dissimilarity between the samples (i.e.: the larger k is, the more dissimilar the sample distributions are) and from that, observe the behaviour of the distribution of the p-value.

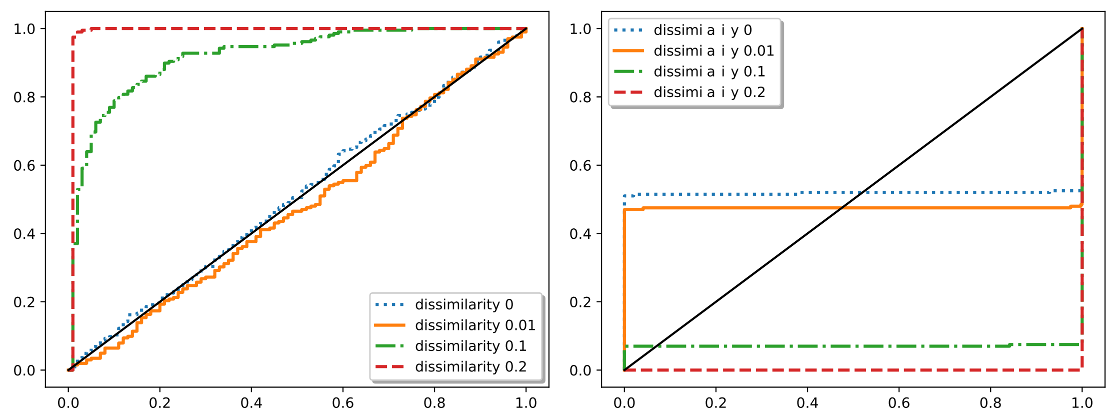
In Figure 14a, we present the results of such experiment using the permutation test (with 100 permutations): the empirical cumulative distribution of the p-values; while in Figure 14b, we do the same simulation study using an Gaussian asymptotic approximate to the permutation test.

The permutation test fulfilled the required properties of a frequentist hypothesis test, as it has approximately (sub)uniform distribution under the null hypothesis (as expected) and the test power increases as the divergence increases. Notice that, for simplicity, we do not use refits here; if we do, we expect the power to increase as is the case in the next section. The asymptotic test, on the other hand, performed poorly.

4.3.2 Evaluation (images)

Here, we also applied the hypothesis testing method to the CIFAR10 dataset, using the same 2500 images for each category as described in 4.2.2. In Tables 5, 6, 7 and 8 we present the p-values obtained in the test while in Tables 9, 10, 11 and 12 we present the combinations that gave the correct results and type 2 error for a significance level of 5%. We applied the tests both without VAE refits and with 5 refits; we also tried the median as an alternative to the mean with the intuition that this might help remove the weight of outlier distance points.

In Table 13, we present a summary of the results. The method performed well under the null for both the mean and median metrics. Moreover, the method has shown to have a significant increase in test power when used with VAE refits.



(a) Permutation. (b) Asymptotic.

Figure 14 – Empirical cumulative distribution function of the p-values for distinct dissimilarity values (when the dissimilarity is zero, the null hypothesis is true) using a permutation test and asymptotic (approximate to permutation test).

In case of metrics performance comparison, it can be seem that the mean had incurred in less type I errors while the median incurred in larger but an admissible number given the critical rate of 5%. On the other hand the performance of median metric was considerably better regarding type II errors, this might be related to its robustness to outliers which have shown to be a frequent problem in the Figure 13.

Table 5 – P-values for hypothesis testing for each category **without** refits and averaging using the **median**.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	0.99	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.40	0.01
c1	-	0.42	0.56	0.01	0.01	0.01	0.05	0.24	0.03	0.37
c2	-	-	0.74	0.50	0.40	0.04	0.58	0.04	0.01	0.01
c3	-	-	-	0.78	0.31	0.26	0.49	0.22	0.01	0.01
c4	-	-	-	-	0.39	0.21	0.29	0.23	0.01	0.01
c5	-	-	-	-	-	0.02	0.01	0.01	0.01	0.01
c6	-	-	-	-	-	-	0.96	0.32	0.01	0.01
c7	-	-	-	-	-	-	-	0.78	0.01	0.01
c8	-	-	-	-	-	-	-	-	0.06	0.01
c9	-	-	-	-	-	-	-	-	-	0.54

4.3.3 Evaluation (comparison with other methods)

Next, we apply our proposed hypothesis testing method (using the median as the averaging function and 10 refits) to simulated datasets from a known data generating function and compare it with other well established two-sample comparison methods: Mann-Whitney rank test (MANN; WHITNEY, 1947), Kolmogorov-Smirnov (SMIRNOV, 1948) and Welch's t-test (WELCH, 1947).

Table 6 – P-values for hypothesis testing for each category **with** refits and averaging using the **median**.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	0.20	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.02	0.01
c1	-	0.26	0.03	0.01	0.01	0.01	0.01	0.01	0.01	0.02
c2	-	-	0.36	0.20	0.05	0.04	0.07	0.01	0.01	0.01
c3	-	-	-	0.41	0.06	0.14	0.25	0.03	0.01	0.01
c4	-	-	-	-	0.90	0.02	0.01	0.02	0.01	0.01
c5	-	-	-	-	-	0.94	0.02	0.01	0.01	0.01
c6	-	-	-	-	-	-	0.02	0.01	0.01	0.01
c7	-	-	-	-	-	-	-	1.00	0.01	0.01
c8	-	-	-	-	-	-	-	-	0.36	0.01
c9	-	-	-	-	-	-	-	-	-	0.03

Table 7 – P-values for hypothesis testing for each category **without** refits and averaging using the **mean**.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	0.25	0.12	0.01	0.01	0.01	0.01	0.01	0.01	0.48	0.05
c1	-	0.82	0.50	0.23	0.03	0.05	0.07	0.15	0.01	0.05
c2	-	-	0.19	0.49	0.48	0.44	0.09	0.02	0.10	0.01
c3	-	-	-	0.22	0.11	0.36	0.15	0.27	0.01	0.01
c4	-	-	-	-	0.87	0.20	0.40	0.01	0.01	0.01
c5	-	-	-	-	-	0.19	0.05	0.23	0.01	0.01
c6	-	-	-	-	-	-	0.73	0.01	0.01	0.01
c7	-	-	-	-	-	-	-	0.48	0.01	0.02
c8	-	-	-	-	-	-	-	-	0.45	0.06
c9	-	-	-	-	-	-	-	-	-	0.48

Table 8 – P-values for hypothesis testing for each category **with** refits and averaging using the **mean**.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	0.16	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.03	0.01
c1	-	0.30	0.07	0.01	0.01	0.01	0.01	0.01	0.01	0.02
c2	-	-	0.10	0.06	0.12	0.37	0.04	0.01	0.01	0.01
c3	-	-	-	0.83	0.08	0.45	0.15	0.02	0.01	0.01
c4	-	-	-	-	0.53	0.05	0.10	0.20	0.01	0.01
c5	-	-	-	-	-	0.50	0.01	0.01	0.01	0.01
c6	-	-	-	-	-	-	0.12	0.05	0.01	0.01
c7	-	-	-	-	-	-	-	0.48	0.01	0.01
c8	-	-	-	-	-	-	-	-	0.18	0.01
c9	-	-	-	-	-	-	-	-	-	0.71

Table 9 – Results of the hypothesis testing when applying a critical rate of 5% **without** refits and averaging using the **median**. Here G stands for “good”, E1 for type 1 error and E2 for type 2 error.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	G	G	G	G	G	G	G	G	E2	G
c1	-	G	E2	G	G	G	G	E2	G	E2
c2	-	-	G	E2	E2	G	E2	G	G	G
c3	-	-	-	G	E2	E2	E2	E2	G	G
c4	-	-	-	-	G	E2	E2	E2	G	G
c5	-	-	-	-	-	E1	G	G	G	G
c6	-	-	-	-	-	-	G	E2	G	G
c7	-	-	-	-	-	-	-	G	G	G
c8	-	-	-	-	-	-	-	-	G	G
c9	-	-	-	-	-	-	-	-	-	G

Table 10 – Results of the hypothesis testing when applying a critical rate of 5% **with** refits and averaging using the **median**. Here G stands for “good”, E1 for type 1 error and E2 for type 2 error.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	G	G	G	G	G	G	G	G	G	G
c1	-	G	G	G	G	G	G	G	G	G
c2	-	-	G	E2	G	G	E2	G	G	G
c3	-	-	-	G	E2	E2	E2	G	G	G
c4	-	-	-	-	G	G	G	G	G	G
c5	-	-	-	-	-	G	G	G	G	G
c6	-	-	-	-	-	-	E1	G	G	G
c7	-	-	-	-	-	-	-	G	G	G
c8	-	-	-	-	-	-	-	-	G	G
c9	-	-	-	-	-	-	-	-	-	E1

Table 11 – Results of the hypothesis testing when applying a critical rate of 5% **without** refits and averaging using the **mean**. Here G stands for “good”, E1 for type 1 error and E2 for type 2 error.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	G	E2	G	G	G	G	G	G	E2	G
c1	-	G	E2	E2	G	G	E2	E2	G	G
c2	-	-	G	E2	E2	E2	E2	G	E2	G
c3	-	-	-	G	E2	E2	E2	E2	G	G
c4	-	-	-	-	G	E2	E2	G	G	G
c5	-	-	-	-	-	G	G	E2	G	G
c6	-	-	-	-	-	-	G	G	G	G
c7	-	-	-	-	-	-	-	G	G	G
c8	-	-	-	-	-	-	-	-	G	E2
c9	-	-	-	-	-	-	-	-	-	G

Table 12 – Results of the hypothesis testing when applying a critical rate of 5% **with** refits and averaging using the **mean**. Here G stands for "good", E1 for type 1 error and E2 for type 2 error.

	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	G	G	G	G	G	G	G	G	G	G
c1	-	G	E2	G	G	G	G	G	G	G
c2	-	-	G	E2	E2	E2	G	G	G	G
c3	-	-	-	G	E2	E2	E2	G	G	G
c4	-	-	-	-	G	G	E2	E2	G	G
c5	-	-	-	-	-	G	G	G	G	G
c6	-	-	-	-	-	-	G	G	G	G
c7	-	-	-	-	-	-	-	G	G	G
c8	-	-	-	-	-	-	-	-	G	G
c9	-	-	-	-	-	-	-	-	-	G

Table 13 – Summary of the results of the hypothesis testing when applying a critical rate of 5%.

Averaging	Refits	Number Type I errors	Number Type II errors
mean	with	0	18
mean	without	0	38
median	with	2	10
median	without	1	30

Given that such methods work only with univariate datasets, we choose the true distribution of the generating data to be a mixture of 3 equiprobable Gaussian distributions with means -2, 0 and 2 and standard deviation 1. For testing the alternative hypothesis, one of the datasets had a 0.1 disturbance added (i.e., the dataset is generated from a mixture of 3 equiprobable Gaussian distributions with means -2.1, 0.1 and 2.1). Each dataset being compared is composed of $n = 1000$ instances generated independently from the true distribution. Due to computational limitations, the power function was estimated using 500 simulations for vaecompare, while we used 10000 simulations for the other tests.

Figure 15 shows the mean test power of each test with a confidence band of 2 times the standard error (i.e., an approximately 95% confidence). In order to make the visualization of the results easier, we also present a smoothed version of this figure in Figure 16 with smoothing done by simple point interpolation[3].

The figures indicate that *vaecompare* had competitive performance when compared to the other methods.[4] Additionally, it is the only method that can be used exploratory

[3] Such smoothed version could also be obtained by increasing the number of permutations for each test, this has not been done due to computational constraints of such increase.

[4] Note in particular, that, contrary to the problems of classification and regression, is not possible to easily data split the dataset to choose the best hypothesis testing method before

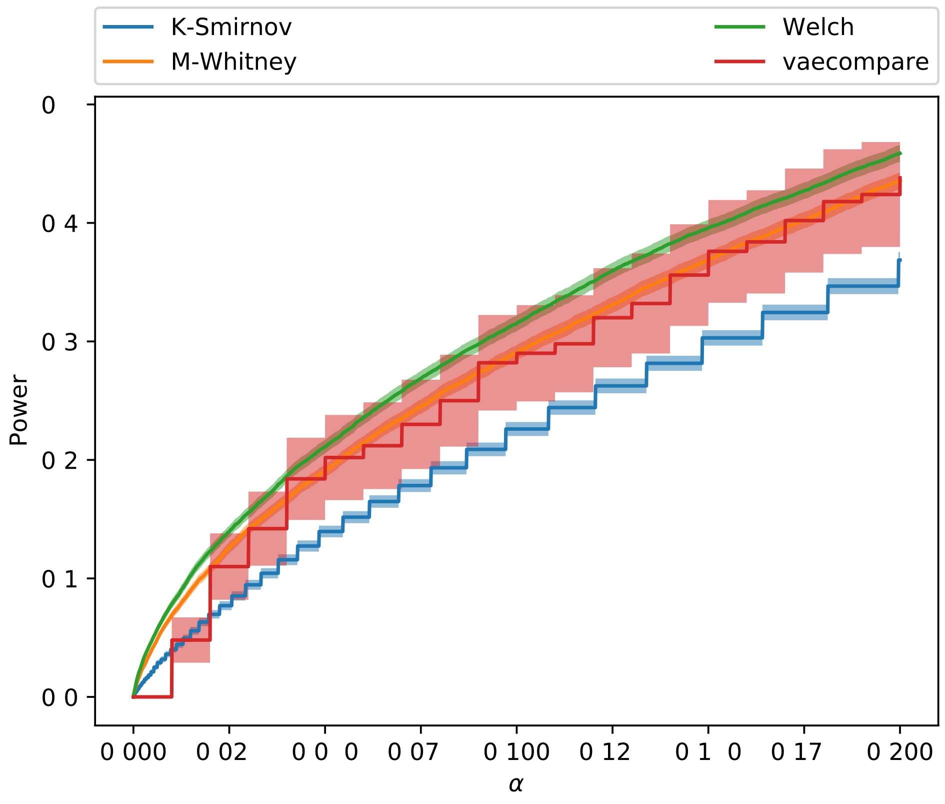


Figure 15 – Comparison of vaecompare with other hypothesis testing methods. Our procedure shows good power.

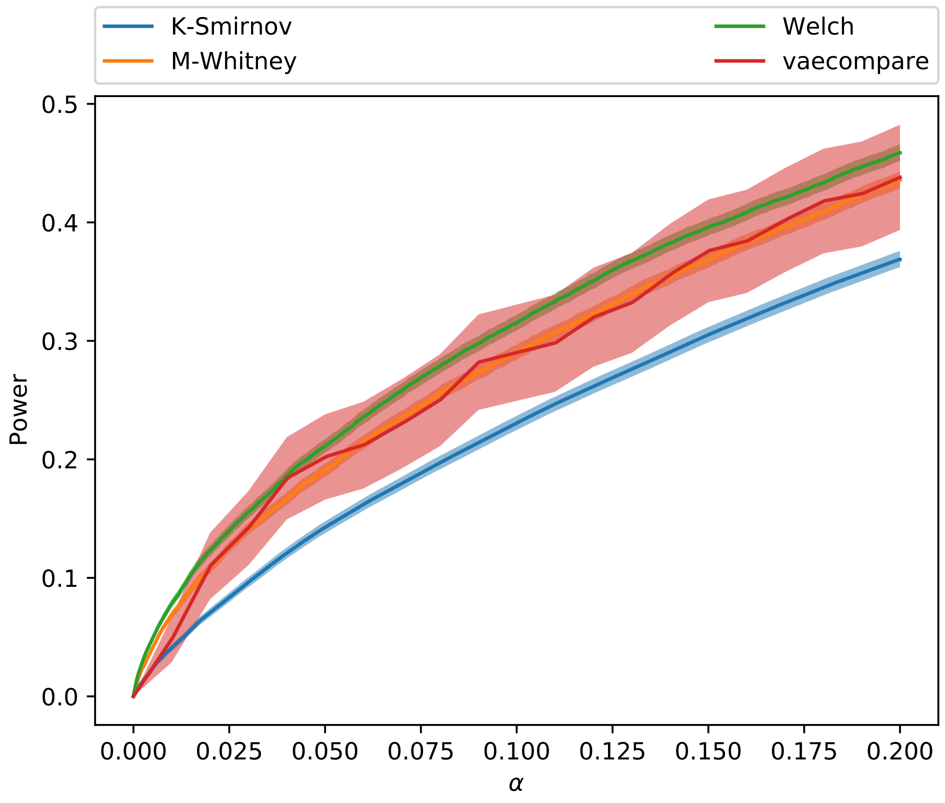


Figure 16 – Comparison of vaecompare with other hypothesis testing methods. Points outside of the grid are smoothed by interpolation. Our procedure shows good power.

data analysis (on two-sample comparison) instead of just hypothesis testing and moreover, as shown in Section 4.3.2, the test power could potentially increase if an additional number of refits, which were set to be a small number because of the computational restrictions of the simulation study.

4.4 Discussion and Conclusions

In this chapter, we proposed and applied a novel method of two sample distance measurement and hypothesis testing to simulated and real-world datasets. We conclude that both two sample distance measurement and hypothesis testing were able to satisfactorily perform the intended tasks on the tested simulated and real world datasets.

The proposed methods could be used for various tasks in the machine learning pipeline, including:

- Distribution shift detection and measurement: a dataset from a experiment done in one month (e.g.: opinions of customers on a product on a specific month) might diverge in distribution from a dataset collected in another month. With our method it is possible to measure and test this diverge.

- Dataset split: to address overfitting, the data is usually split into train, validation and/or test parts. To be able to develop robust models, these parts should be similar, but should also differ enough to ensure generalization. With the proposed methods the dataset split can be done in a controlled manner, an important speed on state-of-the-art predictive methods (e.g.: see Breiman (1996), Coscrato, Inácio and Izbicki (2020) and references therein).

- Self-supervised clustering: based on the distance, by fine-tuning the threshold (cut-point), binary or multi-class clustering could be performed.

- Anomaly detection: applying the proposed method to processes where anomaly may occur (e.g. malicious attack, malfunction, etc.). In this case, the distance measurement can give a direct feedback of how much the actual behaviour differs from the normal one.

- To test the quality of data generated from GANs and similar approaches (e.g.: see Lopez-Paz and Oquab (2017)).

applying it the whole dataset (at least not without causing further problems such as bias multiple comparisons).

Appendix: Neural networks configuration, software and package

We work with a dense neural network of 10 layers with 100 neurons on each layer (totaling 195060 parameters), for both encoder and decoder networks, and the following additional specification:

- **Optimizer**: we work with the Adamax optimizer (KINGMA; BA, 2014) with initial learning rate of 0.01 and decrease its learning rate by half if no improvement is seen on the validation loss for a considerable number of epochs.
- **Initialization**: we used the initialization method proposed by (GLOROT; BENGIO, 2010a).
- **Layer activation**: we chose ELU (DJORK-ARNÉ; UNTERTHINER; HOCHREITER, 2016) as activation functions.
- **Stop criterion**: a 90%/10% split early stopping for small datasets and a higher split factor for larger datasets (increasing the proportion of training instances) and a patience of 50 epochs without improvement on the validation set.
- **Normalization**: batch normalization, as proposed by (IOFFE; SZEGEDY, 2015), is used in this chapter in order to speed-up the training process.
- **Dropout**: here we also make use of dropout which as proposed by (SRIVASTAVA *et al.*, 2014) (with dropout rate of 0.5).
- **Software**: we have PyTorch(PASZKE *et al.*, 2019) as framework of choice which works with automatic differentiation and the sstudy Python package(INÁCIO, 2020) for organizing the simulation studies and comparisons. Moreover, the software implementation of this chapter is available at <https://github.com/randommm/vaecompare>.

Additionally, we present in algorithms 4 and 5 the algorithm to evaluate the encoder and decoder neural networks, respectively.

Algorithm 4 – Algorithm to evaluate encoder network g(.) presented in Figure 11

Input: x,

Output: μ, σ.

```
 1: val = x
 2: for $i \in \{1, 10\}$ do
 3:     val = linear(val) (with output size 100).
 4:     val = ELU(val).
 5:     val = batch_norm(val)
 6:     val = dropout(val)
 7: end for
 8: $\mu$ = linear(val)
 9: $\sigma$ = exp{linear(val)}.
```

Algorithm 5 – Algorithm to evaluate decoder network h(.) presented in Figure 11

Input: z, distribution

Output: (μ, σ) or p.

```
 1: val = z
 2: for $i \in \{1, 10\}$ do
 3:     val = linear(val) (with output size 100).
 4:     val = ELU(val).
 5:     val = batch_norm(val)
 6:     val = dropout(val)
 7: end for
 8: if distribution is ”gaussian” (i.e. continuous data) then
 9:     $\mu$ = linear(val)
10:     $\sigma$ = exp{linear(val)}.
11: else if distribution is ”bernoulli” then
12:     $p$ = sigmoid{linear(val)}.
13: end if
```

CHAPTER

5

CONDITIONAL INDEPENDENCE TESTS: A PREDICTIVE PERSPECTIVE

Conditional independence testing is required by many methods in machine learning and statistics, including Bayesian networks (JENSEN, 1996; CAMPOS, 2006), time series (DIKS; PANCHENKO, 2006), causal inference (SPIRTES *et al.*, 2000; PEARL, 2009) and feature selection (KOLLER; SAHAMI, 1996). However, it is not possible to design conditional independence tests that are powerful against all points in the alternative hypothesis (SHAH; PETERS, 2018). This issue is partially addressed by making assumptions about the data distribution. Indeed, various conditional independence methods have a high power for alternative hypotheses of interest (DORAN *et al.*, 2014; SEN *et al.*, 2017; BERRETT *et al.*, 2018; CHALUPKA; PERONA; EBERHARDT, 2018).

This chapters develops conditional independence testing to evaluate the usefulness of features on a prediction problem. More precisely, let $\mathbf{X} = (X_1, \ldots, X_p)$ be a set of features, $\mathbf{X}_A$ be a subset of $\mathbf{X}$, and $\mathbf{X}_O$ be the remaining features. Our goal is to test whether $\mathbf{X}_A$ is independent of the label, Y, given $\mathbf{X}_O$, that is, whether $H_0 : \mathbf{X}_A \perp Y | \mathbf{X}_O$ holds. Although testing H_0 is related to *measures of feature importance* (BREIMAN, 2001; STROBL *et al.*, 2008; FISHER; RUDIN; DOMINICI, 2018), they are not the same. While the latter quantifies how informative is a given feature, the former answers whether or not the feature is relevant.

In order to test H_0, this chapter investigates permutation tests (GOOD, 2013). Specifically, we propose tests that compare the performance of a predictive method using the actual data and new sets in which the values of $\mathbf{X}_A$ are permuted. Note that the permuted values of $\mathbf{X}_A$ are not informative for predicting Y. The better the predictive method performs on the actual data over the permuted data, the more evidence there is against H_0. Although the test in Watson and Wright (2019a) also compares the risk of two prediction methods, it is not based on permutations of the data. We show that, specially for small sample sizes, such permutation tests can approximately control the significance level while achieving higher power against relevant alternative hypotheses.

The remaining of this chapter is organized as follows. Section 5.1 introduces COINP, our approach to test conditional independence. Section 5.2 contains experiments for comparing COINP with other approaches while Section 5.3 presents an illustrative example of applying the method to a real world dataset together the classical importance measure obtained from random forests.

5.1 Predictive conditional independence tests

5.1.1 Notation and problem setting

Let $\mathscr{X} = \mathbb{R}^p$, $\mathscr{Y}$, and $\mathscr{Z} = (\mathscr{X} \times \mathscr{Y})^n$ denote, respectively, the feature space, the label space and the space of all possible datasets. The observed data is $\mathbb{Z} = (\mathbb{X}, \mathbf{Y})$, where $\mathbb{X} \in \mathscr{X}^n$ is a $n \times p$ feature matrix and $\mathbf{Y} \in \mathscr{Y}^n$ is the label vetor. We assume that the observations $\mathbf{Z}_i = (\mathbf{X}_i, Y_i)$, $i = 1, \ldots, n$, are independent and identically distributed. Let $A \subset \{1, \ldots, p\}$, $B = A^c$, and $\mathbf{X}_A = (X_i)_{i \in A}$ be a subset of the features. Our goal is to test whether $\mathbf{X}_A$ is independent of Y given $\mathbf{X}_O$, the remaining variables, that is, we have $H_0 : \mathbf{X}_A \perp Y | \mathbf{X}_O$.

In order to test H_0, we use the estimated risk of predictive functions. A predictive function is a mapping from features to labels, that is, an element of $\mathscr{F} := \{f : \mathscr{X} \longrightarrow \mathscr{Y}\}$. Predictive functions are compared according their risks. Given a loss functions, $L : \mathscr{Y} \times \mathscr{Y} \longrightarrow \mathbb{R}$, the risk of a prediction function $f \in \mathscr{F}$ is $R(f) := \mathbb{E}[L(f(\mathbf{X}), Y)]$. The Bayes predictive function, f^* is the one with the smallest risk, that is, $f^* = \arg\min_{f \in \mathscr{F}} R(f)$. The risk is estimated using a holdout dataset, $\tilde{\mathbb{Z}} \in (\mathscr{X} \times \mathscr{Y})^m$, which was not used for finding f. Formally, the estimated risk of f, $\widehat{R}(f, \tilde{\mathbb{Z}})$, is

$$\widehat{R}(f, \tilde{\mathbb{Z}}) := \frac{1}{m} \sum_{i=1}^{m} L(f(\tilde{\mathbf{x}}_i), \tilde{y}_i).$$

The **Co**nditional **In**dependence **P**redictive Test (COINP) uses the intuition that, if $\mathbf{X}_A \perp Y | \mathbf{X}_O$, then the Bayes predictive function based solely on $\mathbf{X}_O$ should have the same risk as the Bayes predictive function based on $\mathbf{X}$. In order to test whether this is the case, a permutation test is used. In this procedure, a prediction function is trained in a dataset that is obtained by randomly permuting the rows of $\mathbb{X}$ associated to the features in A. This procedure is illustrated in Figure 17 when $A = \{3\}$. Formally, for every $j = 1, \ldots, B$, π_j denotes the j-th permutation function, $\mathbb{Z}^{\pi_j}$ denotes the j-th permuted dataset, $\mathbf{x}_i^{\pi_j}$ denotes the i-th feature row of the permuted dataset, and $\tilde{\mathbb{Z}}^{\pi_j}$ denotes the j-th permuted holdout set.

Table 14 summarizes the above notation.

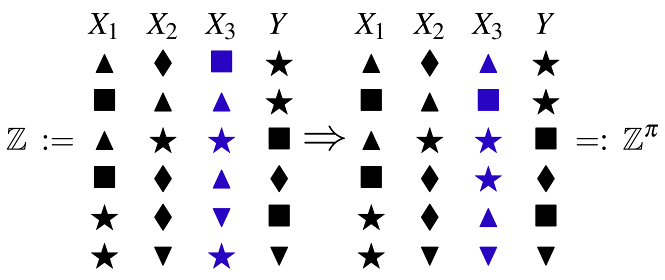


Figure 17 – Illustration of the procedure that is used for testing $X_3 \perp Y | (X_1, X_2)$. For each permutation function, π, the permuted database, $\mathbf{Z}^{\pi}$ is obtained by permuting the feature X_3 in $\mathbf{Z}$ by π.

Table 14 – Notation used in the chapter.

Symbol	Meaning
$\mathbb{X}$	$n \times p$ feature matrix
Y	label vector
$\mathbb{Z}$	training data
$\tilde{\mathbb{Z}}$	holdout data
$\mathscr{F}$	space of prediction functions
$\widehat{R}(f, \mathbb{Z})$	risk estimate of $f \in \mathscr{F}$ based on $\mathbb{Z}$
$\mathbb{Z}^{\pi_j}$	dataset in which the A columns of $\mathbb{Z}$ are permuted by π_j
$\mathbf{x}_i^{\pi_j}$	features of the i-th row of $\mathbb{Z}^{\pi_j}$

5.1.2 Conditional independence predictive test (COINP)

The COINP test is based on comparing the estimated risk of a prediction method on the actual data and on permuted data. A prediction method is a function that assigns a mapping in $\mathscr{F}$ to each possible dataset, that is, a function in $\mathscr{A} := \{\hat{f} : \mathscr{Z} \longrightarrow \mathscr{F}\}$. Formally, the p-value associated COINP is obtained by calculating the estimated risks of a prediction method on the actual data and on permuted data and computing the proportion of permuted data that have a risk lower than that on the actual data. That is, the p-value given by $\hat{f} \in \mathscr{A}$ is

$$p_{COINP} := \frac{1}{B} \sum_{j=1}^{B} \mathbb{I}\left(\widehat{R}(\hat{f}(\mathbb{Z}), \tilde{\mathbb{Z}}) \geq \widehat{R}(\hat{f}(\mathbb{Z}^{\pi_j}), \tilde{\mathbb{Z}}^{\pi_j})\right). \tag{5.1}$$

Intuitively, if H_0 does not hold, then even after $\mathbf{X}_O$ is observed, $\mathbf{X}_A$ still provides more information about Y. Therefore, since permuting the rows of the features in A transforms them into noise, the estimated risks of a prediction method based on the permuted data will probably be larger than that in the actual data. Similarly, if H_0 holds, then the features in $\mathbf{X}_O$ have all the relevant information for predicting Y. Therefore, the estimated risk for the prediction method based on the actual data and on the permuted data should be similar. The COINP is also described in Algorithm 6.

Algorithm 6 – COINP

Input: training data $\mathbb{Z}$, testing data $\tilde{\mathbb{Z}}$, prediction method $\hat{f} \in \mathscr{A}$, loss L, feature indices A, number of simulations B

Output: p-value for testing $H_0 : \mathbf{X}_A \perp Y | \mathbf{X}_O$

```
1: $f \leftarrow \hat{f}(\mathbb{Z})$
2: $R \leftarrow \widehat{R}(f, \tilde{\mathbb{Z}})$
3: for $j \in \{1, \dots, B\}$ do
4:     Compute $\mathbb{Z}^{\pi_j}$ by randomly permuting the columns of $\mathbb{Z}$ associated to features in $A$
5:     $f_j \leftarrow \hat{f}(\mathbb{Z}^{\pi_j})$
6:     Compute $\tilde{\mathbb{Z}}^{\pi_j}$ by randomly permuting the columns of $\tilde{\mathbb{Z}}$ associated to features in $A$
7:     $R_j \leftarrow \widehat{R}(f_j, \mathbb{Z}^{\pi_j})$
8: end for
9: return $(|\{j : R \geq R_j\}| + 1)/(B + 1)$
```

5.1.3 Other approaches to predictive conditional independence testing

5.1.3.1 Approximate COINP (ACOINP)

A drawback of COINP occurs when $\hat{f}$ is a computationally intensive predictive method, since $\hat{f}$ is applied to every permuted dataset. One might try to overcome this hindrance by defining

$$p_{ACOINP} := \frac{1}{B} \sum_{j=1}^{B} \mathbb{I}\left(\widehat{R}(\hat{f}(\mathbb{Z}), \tilde{\mathbb{Z}}) \geq \widehat{R}(\hat{f}(\mathbb{Z}), \tilde{\mathbb{Z}}^{\pi_j})\right). \tag{5.2}$$

In order to obtain p_{ACOINP}, the predictive method, a, is solely applied to the actual data. That is, CPI is calculated by applying Algorithm 6, with the exception line 4 and 5 are replaced by $f_j \leftarrow f$.

5.1.3.2 Simple Conditional Predictive Impact (SCPI)

The Simple Conditional Predictive Impact (SCPI) (WATSON; WRIGHT, 2019b) is obtained by training a predictive method once on the actual data and once on permuted data: $f_0 = \hat{f}(\mathbb{Z})$ and $f_1 = \hat{f}(\mathbb{Z}^{\pi_1})$. SCPI tests whether $H_0 : R(f, \tilde{\mathbb{Z}}) \geq R(f_1, \tilde{\mathbb{Z}}^{\pi_1})$ holds. One possible approach to this goal is to define $D_i = L(f_0(\tilde{\mathbf{x}}_i) - L(f_1(\tilde{\mathbf{x}}_i^{\pi_1}))$ and apply a t-test to the hypothesis that the populational mean of D_i is greater or equal than 0. That is, by letting $\bar{D}$ and S_D be the sample mean and standard deviation of D and Φ be the cumulative distribution function of a T distribution with $m-1$ degrees of freedom,

$$p_{SCPI} = \Phi\left(\bar{D} \cdot S_D^{-1}\right) \tag{5.3}$$

5.1.3.3 Approximate SCPI (ASCPI)

Similarly to ACOINP, one could make SCPI less computationally intensive by training the prediction method once only. The approximate SCPI (ASCPI) is obtained by defining $f_0 = \hat{f}(\mathbb{Z})$ and $D_i = L(f_0(\tilde{\mathbf{x}}_i) - L(f_0(\tilde{\mathbf{x}}_i^{\pi_1}))$. A p-value is obtained by applying a t-test to the hypothesis that the populational mean of D_i is greater or equal than 0. This procedure is essentially the same as described by Breiman and Cutler (2008) to obtain p-values for the importance measures produced by random forests, with the exception that Breiman and Cutler (2008) uses a z-test instead of a t-test.

5.1.3.4 Conditional Predictive Impact (CPI)

Conditional Predictive Impact (CPI) (WATSON; WRIGHT, 2019a) is an improvement over SCPI. CPI uses knockoffs instead of the permutations in SCPI.

5.2 Comparison between methods

In order to compare the methods in Section 5.1, they were applied to several simulated and real data. Three prediction methods were used for the prediction methods used in the conditional independence tests: linear regression (LINEAR), random forests (RF), and feedforward neural networks (ANN). The Python implementations are available at <https://github.com/randommm/nnperm>. Both LINEAR and RF were implemented using the scikit-learn Python package (PEDREGOSA *et al.*, 2011). RF ran with default turning parameters, except for the number of trees, which was increased to 300 for better prediction performance. ANN was implemented in PyTorch (PASZKE *et al.*, 2019) with the Adamax optimizar (KINGMA; BA, 2014), the initialization method in Glorot and Bengio (2010a), 5 hidden layers with 100 nodes in each and ELU (DJORK-ARNÉ; UNTERTHINER; HOCHREITER, 2016) as the layer activation. The training process used batch normalization (IOFFE; SZEGEDY, 2015), dropout (SRIVASTAVA *et al.*, 2014), a patience of 50 epochs without improvement in the validation set and a 90%/10% split early stopping for small datasets and a higher split factor for larger datasets. Each ANN was retrained 3 times and the best one was chosen based on performance on the early stopping holdout dataset. The experiments themselves were organized and run using the sstudy Python package (INÁCIO, 2020).

Subsection 5.2.1 compares the type I and II errors of the conditional independence tests in simulated data. Subsection 5.3 applies these tests to real data.

5.2.1 Comparison through simulation

For each setting we run at least 150 independent tests to estimate the power of each test. Additionally, we set $B = 100$ in Algorithm 6.

This sections presents the simulations that were employed to test conditional independence tests. In all of the simulations, the squared error loss, $L(y,\widehat{y}) = (y-\widehat{y})^2$, is used. Also, in all simulations, the hypothesis that was tested was $H_0 : X_1 \perp Y | (X_2,\ldots,X_d)$, that is, $\mathbf{X}_A = X_1$ and $\mathbf{X}_O = \{X_2,\ldots,X_d\}$. The simulations to test can be broken into two scenarios. While in the first scenario the stronger hypothesis that $H_0 : \mathbf{X}_A \perp (Y,\mathbf{X}_O)$ holds, in the second scenario only $H_0 : \mathbf{X}_A \perp Y | \mathbf{X}_O$ holds.

In the first scenario, the target variable is such that $\mathbf{Y} = \mathbf{Z}\beta + \varepsilon$, where $\beta = (0.7, 0.16, 0.39, \beta_A, 0.75)$, $\varepsilon \sim SKN(-0.3, 1.1, 2)$, the skew-normal distribution (AZZALINI; VALLE, 1996), and $\mathbf{Z}$ is a vector of latent variables. These latent variables were generated according two settings of transformations that guarantee that $\mathbf{X}_A \perp (Y, \mathbf{X}_B)$ when $\beta_A = 0$ and that the features and target are heavy-tailed and skewed random variables. Specifically, let $g : \mathbb{R}^4 \to \mathbb{R}^5$, $g(a_1,a_2,a_3,a_4) = (|a_1|^{1.3}, \cos(a_2), \log(|a_1,a_3|), \log(|a_3|), |a_4|^{0.5})$ be an example of a non-linear function. In the first setting of transformation (Distribution 1), $\mathbb{W}$ is a $n \times 4$ matrix such that $W_{i,j} \sim SKN(0, 0.1, 2)$ are i.i.d., $\mathbf{Z}_{i,} = g(\mathbb{W}_{i,})$, and $\mathbb{X}$ is a $n \times 2$ matrix such that $\mathbb{X}_{i,j} = Z_{,j}$. In the second setting (Distribution 2), $\mathbb{X}$ is a $n \times 4$ matrix such that $X_{i,j} \sim SKN(0, 0.1, 2)$ are i.i.d., and $\mathbf{Z} = g(\mathbb{X})$.

In the second scenario, the target variable follows a linear model such that $\mathbf{Y} = \mathbb{X}\beta + \varepsilon$, where $\beta = (3, \beta_A)$. However, when $\beta_A = 0$, contrary to the first scenario in which $\mathbf{X}_A \perp (Y, \mathbf{X}_B)$ holds, in the second scenario only the weaker condition $X_A \perp Y | X_B$ holds. This weaker condition is obtained by generating correlated features according to two settings of distribution. In the first setting (Distribution 3), $\mathbb{X}$ is a $n \times 2$ matrix such that $\mathbb{X} \sim N(0, \Sigma)$, $\Sigma_{i,i} = 1$, $\Sigma_{0,1} = 0.9$ and $\varepsilon \sim N(0, 0.5)$. In the second setting (Distribution 4), $W_{i,j} \sim \text{Beta}(1,1)$ are i.i.d., $Z_i \sim N(-0.5, 1)$, $\mathbb{X}_{i,j} = Z_i + W_{i,j}$ and $\varepsilon \sim \text{Beta}(2,2)$.

Figure 18 describes the empirical cumulative distribution function of the p-value obtained from each method when $n = 1{,}000$ and $H_0 : \mathbf{X}_A \perp \mathbf{Y} | \mathbf{X}_B$ holds, that is, $\beta_A = 0$. Under these circumstances a p-value has the nominal significance when the cumulative distribution function is close to the 45^o degree straight line. From a visual inspection, one can observe that most p-values have are close to the nominal significance under Distributions 1 and 2. However, under Distribution 3 and 4 only CPI and COINP are close to the nominal significance in all scenarios except for LINEAR in Distribution 4. These observations might be explained by two facts. First, ACOINP, SCPI and ASCPI are tests for the stronger hypothesis test $H_0^* : \mathbf{X}_A \perp (\mathbf{Y}, \mathbf{X}_B)$ and, therefore, have incorrect significance whenever $H_0 : \mathbf{X}_A \perp Y | \mathbf{X}_B$ holds but H_0^* does not. This exception can occur when $\mathbf{X}_A$ and $\mathbf{X}_B$ are not independent. Second, under Distribution 4, no test obtains nominal

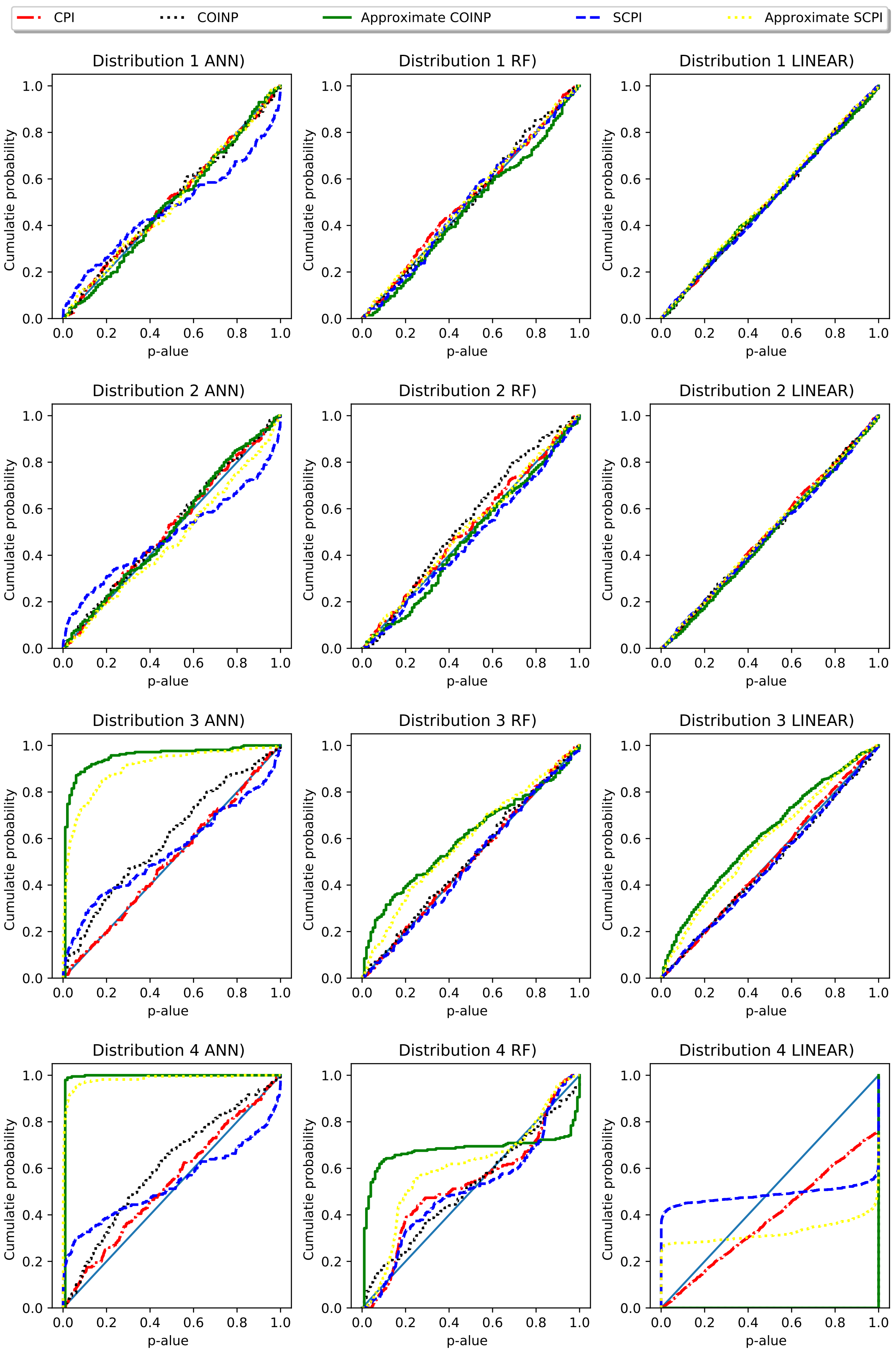


Figure 18 – Cumulative distribution function of the p-values when $n = 1,000$ under H_0.

significance under LINEAR. This fact might be explained by both COINP and CPI being asymptotic tests that require the prediction method to approximate the Bayes regression function when the sample size increases. Qualitatively similar conclusions are obtained when $n = 10,000$, as depicted by figures presented in the supplementary material.

For all the combinations described above, we vary β_S in $\{0, 0.01, 0.1, 0.6\}$ and the number of observations in $\{1000, 10000\}$. Notice that, in all settings, H_0 holds if, and only if, $\beta_A = 0$. Moreover, as $|\beta_A|$ increases, the conditional dependency of Y on $\mathbf{X}^S$ also increases.

Next, we compare the power function of the testing methods. Because only COINP, SCPI and CPI (in most cases) had valid p-values, we restrict the comparisons to these methods. Figures 19 and 20 show[1] the power of each test as a function of β_A. The plot indicates that all procedures achieve higher power as β_A increases. Moreover, in most settings COINP leads to better power than SCPI. In these examples, higher power is achieved when using a linear regression for COINP. This can be explained by the fact that in all settings the true nature of the conditional distribution of $Y|\mathbf{x}$ is close to linear.

On the other hand, when comparing COINP with CPI, we see that performance of both are competitive against each other, with COINP having the advantage of having more stable uniformity under the null as previously presented. This indicates that the testing procedure is consistent and present great balance between uniformity under the null and power under the alternative for the tested settings.

Additionally, in the supplementary material, we present figures repeating this experiment with a larger sample size in order to show that as the sample size increases, so does the test power.

5.2.2 Non-linear distributions

In the previous subsection, we restricted our analysis with simulated datasets that had a linear structure (even though for distributions 3 and 4, we had a correlation structure on the features). Here we analyse two additional distributions that present a non-linear structure. The first non-linear setting (Distribution 5) assumes that $Y = \beta_A X_1 + X_2 + \varepsilon$, with $\varepsilon \sim \text{Gaussian}(0,1)$, $X_1 \sim \text{Gaussian}(0,1)$ and $X_2 = (X_1)^2$. In the last setting (Distribution 6), we take $Y \sim \beta_A * X_1 * X_2 + (X_2)^2 + \varepsilon$, with $\varepsilon \sim \text{Gaussian}(0,1)$, $X_1 \sim \text{Gaussian}(0,1)$ and $X_2 \sim \text{Gaussian}(0,1)$.

In Figures 21-24 we present the empirical cumulative distribution functions of the same experiment done before[2]. For distribution 5, random forests-based were excessively

[1] Additionally, in the supplementary material, we present figures repeating this experiment with a larger sample size in order to show that as the sample size increases, so does the test power.

[2] We plot the whole cumulative distribution function under the alternative hypothesis instead

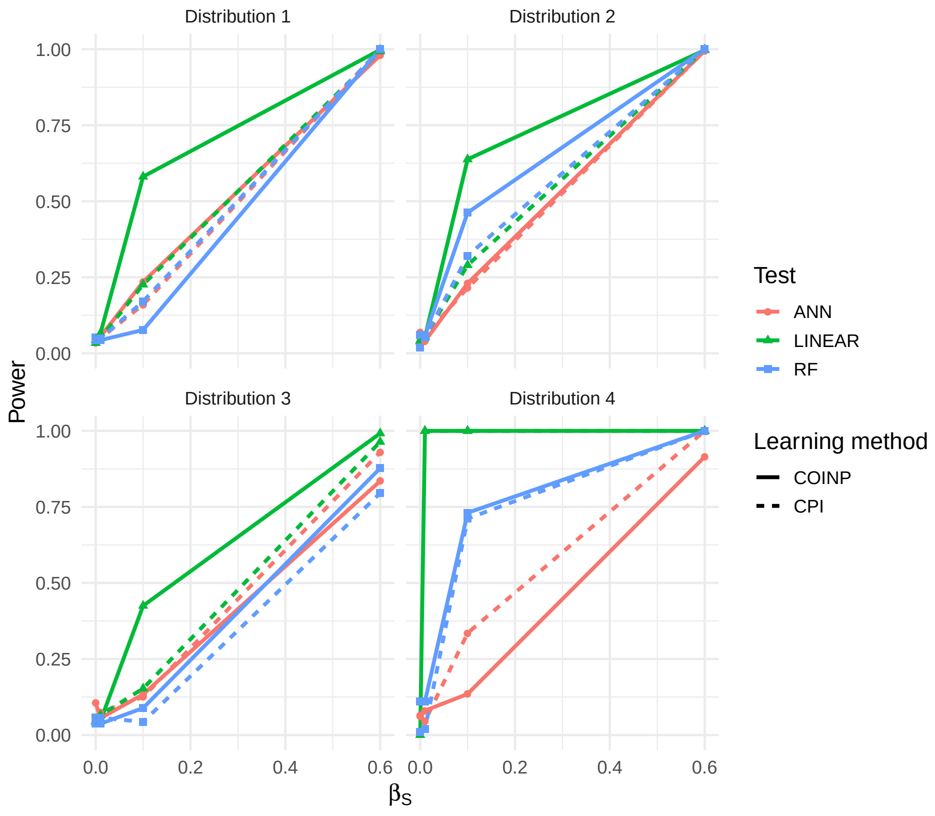


Figure 19 – Power function for all setting with $n = 1{,}000$ for $\alpha = 5\%$ comparing COINP and CPI.

conservative (i.e., p-values were subuniform under the null), which affected its power under the alternative. On the other hand, linear regression had again good performance for distribution 5, but had no power under distribution 6; its cumulative distribution was uniform under the alternative hypothesis. Neural networks has better power for Distribution 6, especially for low signal (i.e.: when the falsifiability of the null hypothesis is less evident).

5.3 A dataset analysis example

In order to create a meaningful comparison of the methods, the ground truth needs to be known. We therefore add artificial covariates to a real dataset to make such comparisons. We use the diamonds dataset available from ggplot2 library (WICKHAM, 2016), and take `price` to be the response variable.

Table 15 presents the p-values for each method using when adding $c+n$, an additional feature generated using the clarity feature plus a standard Gaussian random

of separate power plots as done in the last subsection. We done so in order to facilitate the visualization of the conclusions, especially regarding the lack of power of the linear regression for Distribution 6.

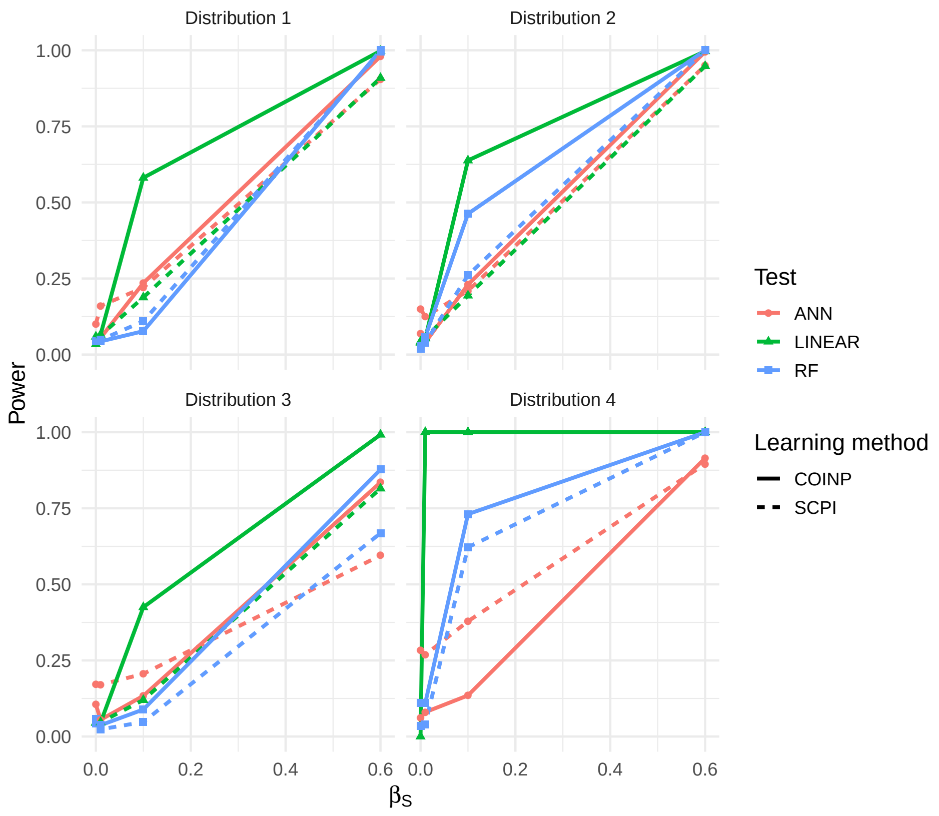


Figure 20 – Power function for all setting with $n = 1{,}000$ for $\alpha = 5\%$ comparing COINP and SCPI.

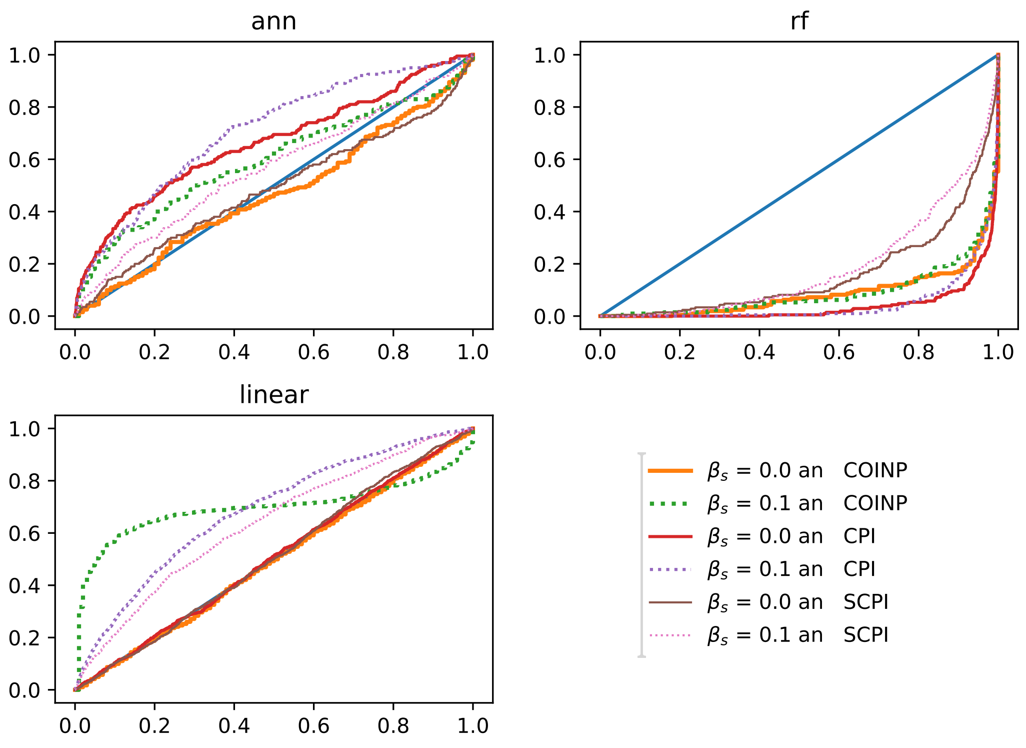


Figure 21 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI for a nonlinear distribution (distribution 5) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.1$ (alternative hypothesis).

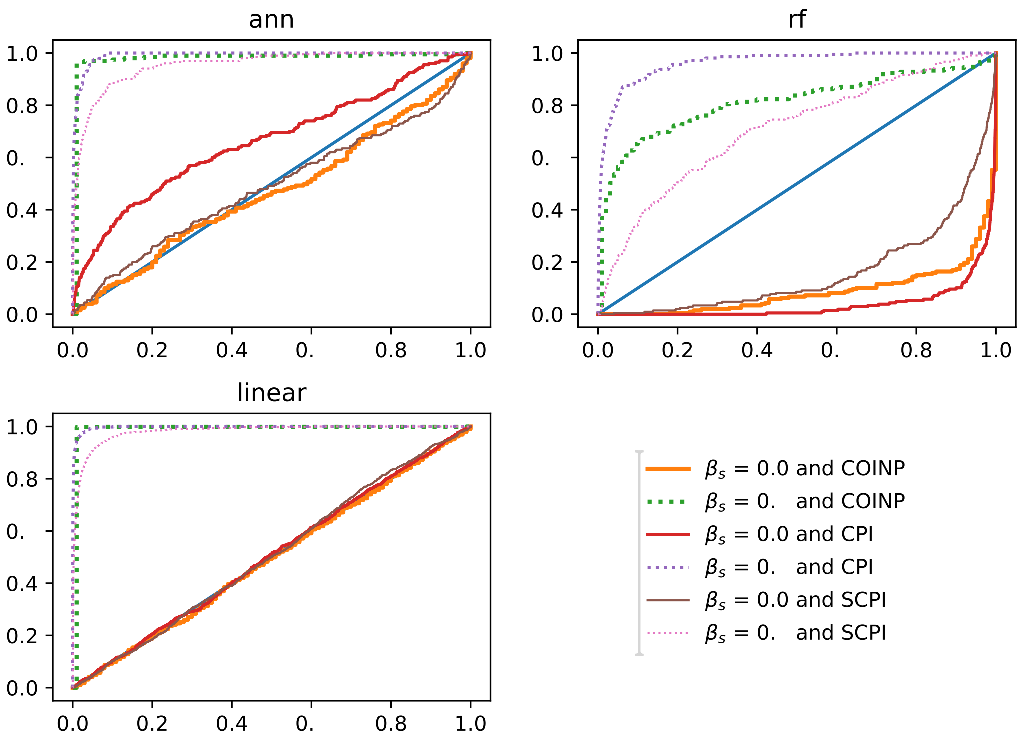


Figure 22 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI for a nonlinear distribution (distribution 5) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.6$ (alternative hypothesis).

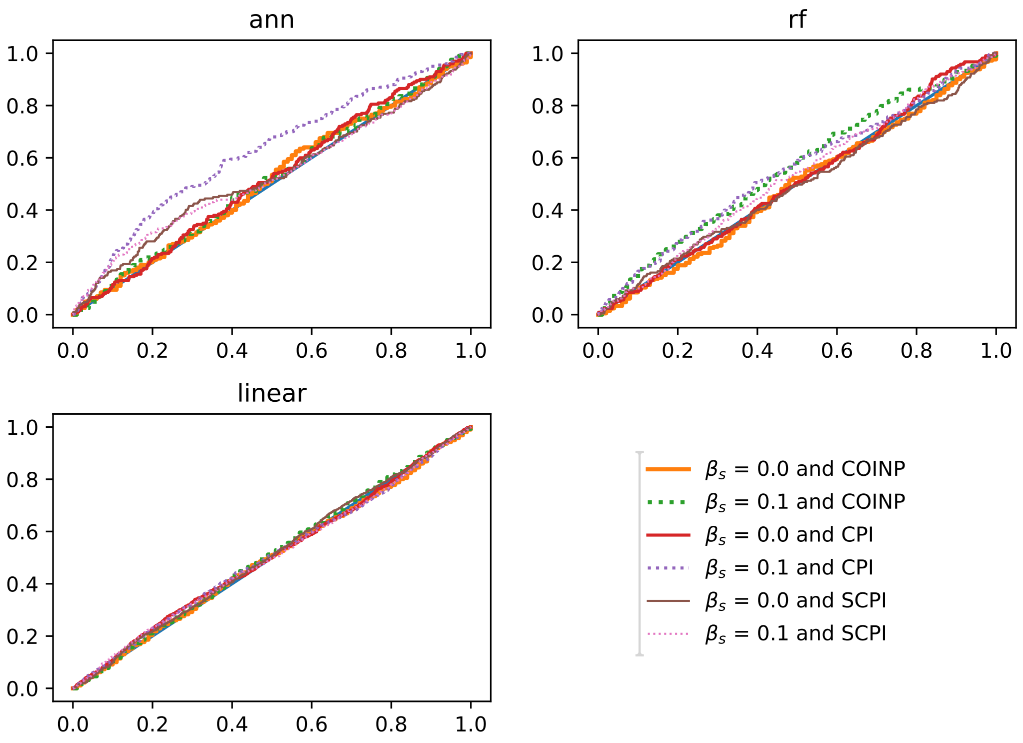


Figure 23 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI for a nonlinear distribution (distribution 6) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.1$ (alternative hypothesis).

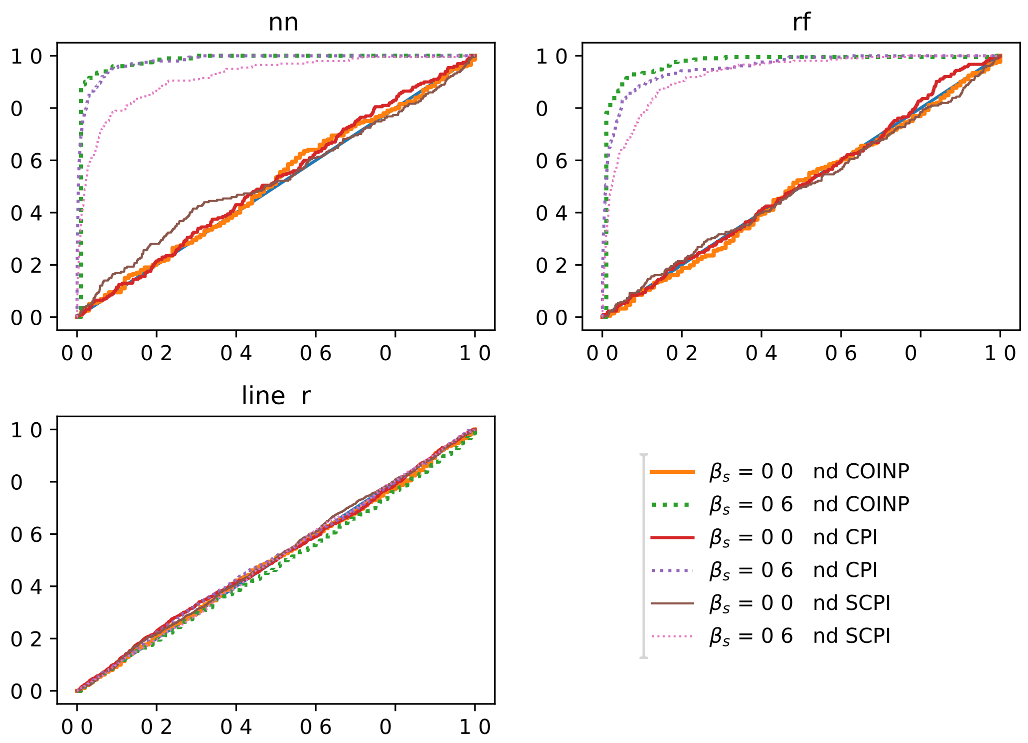


Figure 24 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI for a nonlinear distribution (distribution 6) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.6$ (alternative hypothesis).

noise. By construction, this feature is conditionally independent from the response given the remaining ones. COINP is capable of detecting that (at the level $\alpha = 5\%$) for every regression method. The same does not happen to the approximate methods.

Similarly, in Table 16 we add a new feature, $c+c+n$, generated from the clarity feature plus the color plus a Gaussian random noise[3]. Again, most methods with the exception of the approximate ones gave high p-values most of the time for this new feature, indicating that they were able to capture conditional independence of it and not reject the null hypothesis.

5.4 Final remarks

We have developed a novel approach for testing conditional independence under a predictive setting. We have shown that the p-values obtained by our approach are proper, and that our hypothesis test has larger power than competing approaches under a variety of settings.

When compared to CPI and SCPI, our approach is especially appealing for small sample sizes, because (i) it does not rely on asymptotic approximations such as those

[3] Those two features are categorical and therefore were independently converted to number ranging from 0 to the number of labels (categories) in order to construct the new feature

Table 15 – P-values for hypothesis testing for each comparison method compared to Random forest traditional importance measures using the original database with an extra feature generated from feature clarity plus a Gaussian noise.

		carat	depth	table	x	y	z	cut	color	clarity	c+n
ann	Approx COINP	0.01	0.01	0.39	0.01	0.01	0.01	0.01	0.01	0.01	0.27
	Approx SCPI	0.00	0.12	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.01
	COINP	0.01	0.49	0.36	1.00	0.38	0.40	0.01	0.01	0.01	0.41
	CPI	0.00	0.15	0.04	0.61	0.00	0.24	0.06	0.00	0.00	0.83
	SCPI	0.17	0.24	0.02	0.00	0.93	0.94	0.12	0.00	0.00	0.16
linear	Approx COINP	0.01	0.01	0.01	0.01	0.91	0.01	0.01	0.01	0.01	0.91
	Approx SCPI	0.00	0.00	0.07	0.00	0.02	0.14	0.00	0.00	0.00	0.86
	COINP	0.01	0.01	0.98	0.01	0.32	0.10	0.01	0.01	0.01	0.58
	CPI	0.00	0.51	0.14	0.60	0.12	0.44	0.00	0.00	0.00	0.50
	SCPI	0.00	0.00	0.18	0.77	0.94	0.44	0.00	0.00	0.00	0.14
rf	Approx COINP	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	Approx SCPI	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	COINP	0.01	0.09	0.14	0.13	0.69	0.01	0.01	0.01	0.01	1.00
	CPI	0.00	0.01	0.00	0.89	0.17	0.17	0.06	0.00	0.00	0.13
	SCPI	0.02	0.09	0.44	0.07	0.01	0.01	0.67	0.00	0.00	0.46

Table 16 – P-values for hypothesis testing for each comparison method compared to Random forest traditional importance measures using the original database with an extra feature generated from feature clarity plus feature color plus a Gaussian noise.

		carat	depth	table	x	y	z	cut	color	clarity	c+c+n
ann	Approx COINP	0.01	0.01	0.07	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	Approx SCPI	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	COINP	0.01	0.20	0.61	0.55	0.23	0.04	0.02	0.01	0.78	0.09
	CPI	0.00	0.01	0.16	0.09	0.38	0.76	0.00	0.00	0.00	0.66
	SCPI	0.03	0.87	0.98	0.00	0.69	0.98	0.00	0.00	0.00	0.84
linear	Approx COINP	0.01	0.01	0.01	0.01	0.12	0.01	0.01	0.01	0.01	0.10
	Approx SCPI	0.00	0.00	0.00	0.00	0.63	0.17	0.00	0.00	0.00	0.47
	COINP	0.01	0.01	0.01	0.01	0.30	0.19	0.01	0.01	0.01	0.68
	CPI	0.00	0.99	0.00	0.06	0.49	0.07	0.01	0.00	0.00	0.04
	SCPI	0.00	0.75	0.22	0.78	0.45	0.90	0.00	0.00	0.00	0.41
rf	Approx COINP	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
	Approx SCPI	0.00	0.03	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	COINP	0.01	0.01	0.01	0.33	1.00	0.25	0.04	0.01	0.01	1.00
	CPI	0.00	0.01	0.01	0.74	0.00	0.65	0.14	0.00	0.00	0.38
	SCPI	0.00	0.47	0.11	0.20	0.10	0.07	0.20	0.00	0.00	1.00

required by the t-test, and (ii) its computational burden is not high in those cases.

Appendix

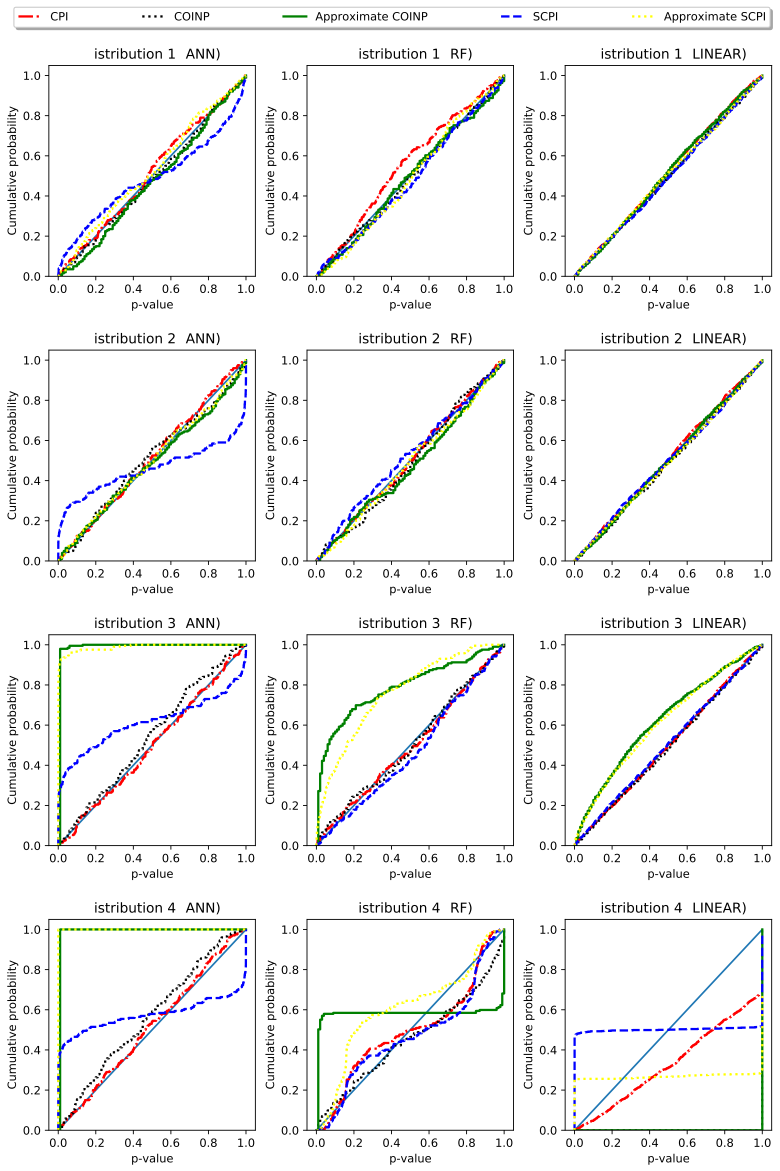


Figure 25 – Cumulative distribution function of the p-values when $n = 10,000$ under H_0.

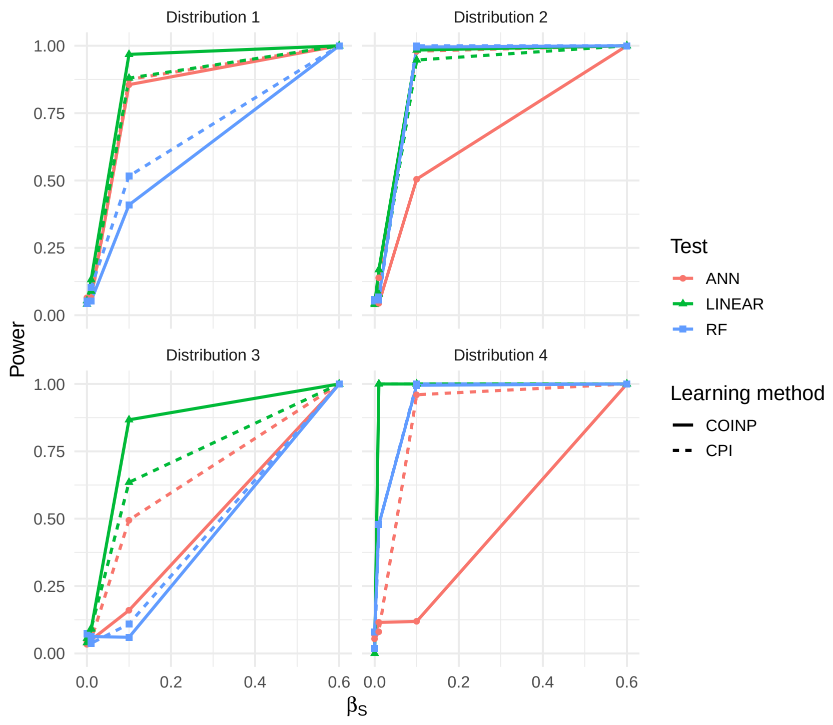


Figure 26 – Power function for all setting with $n = 10,000$ for $\alpha = 5\%$ comparing COINP and CPI.

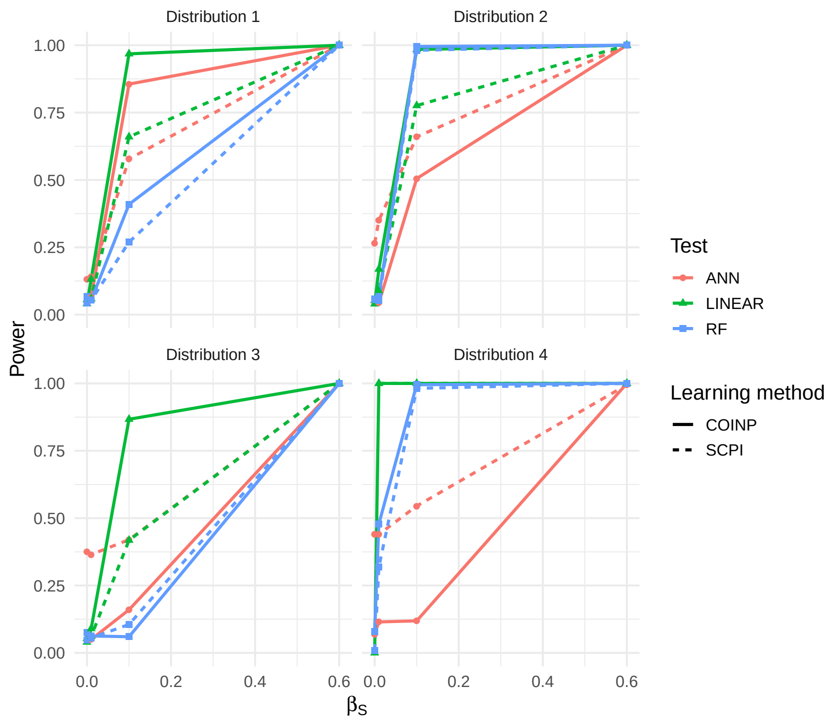


Figure 27 – Power function for all setting with $n = 10,000$ for $\alpha = 5\%$ comparing COINP and SCPI.

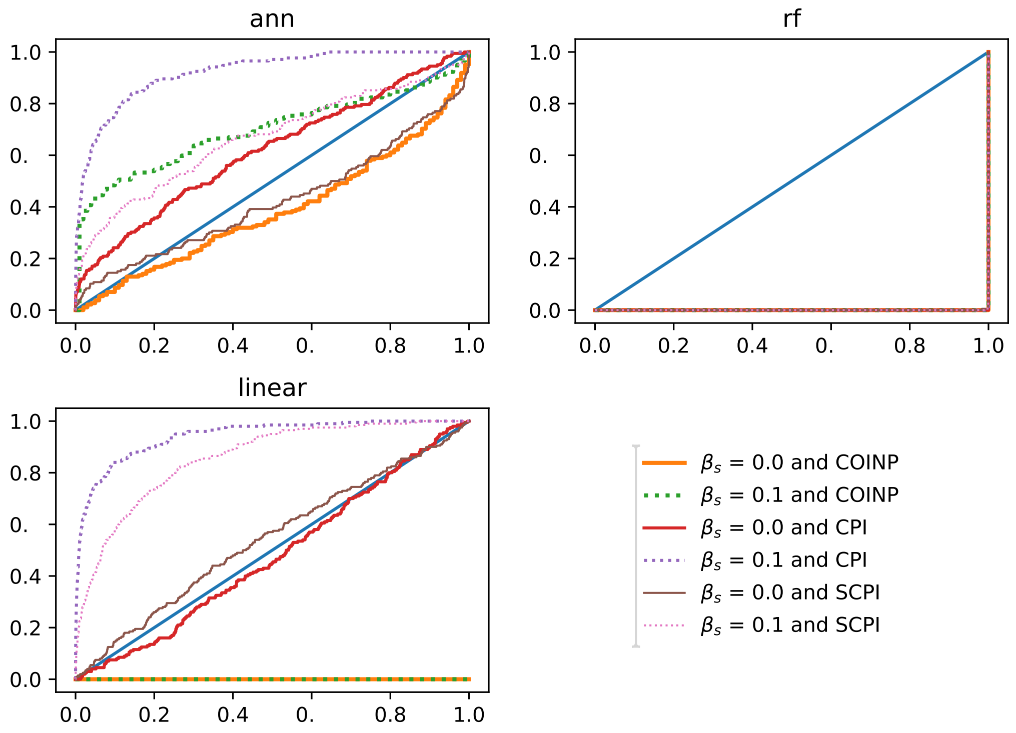


Figure 28 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI with $n = 10,000$ for a nonlinear distribution (distribution 5) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.1$ (alternative hypothesis).

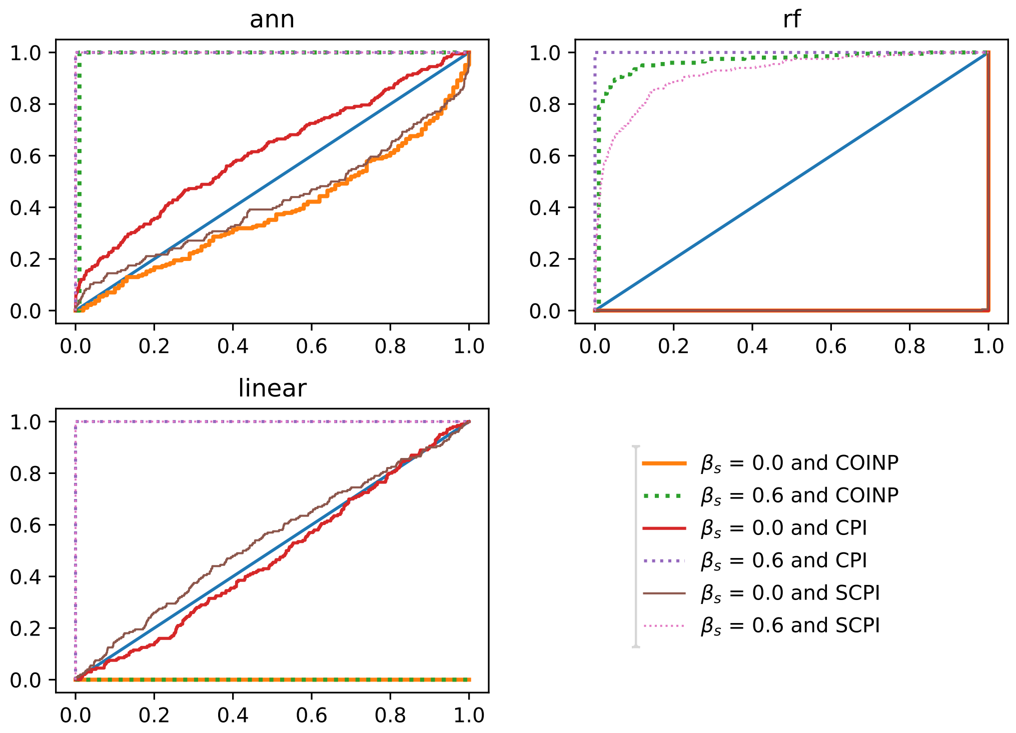


Figure 29 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI with $n = 10,000$ for a nonlinear distribution (distribution 5) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.6$ (alternative hypothesis).

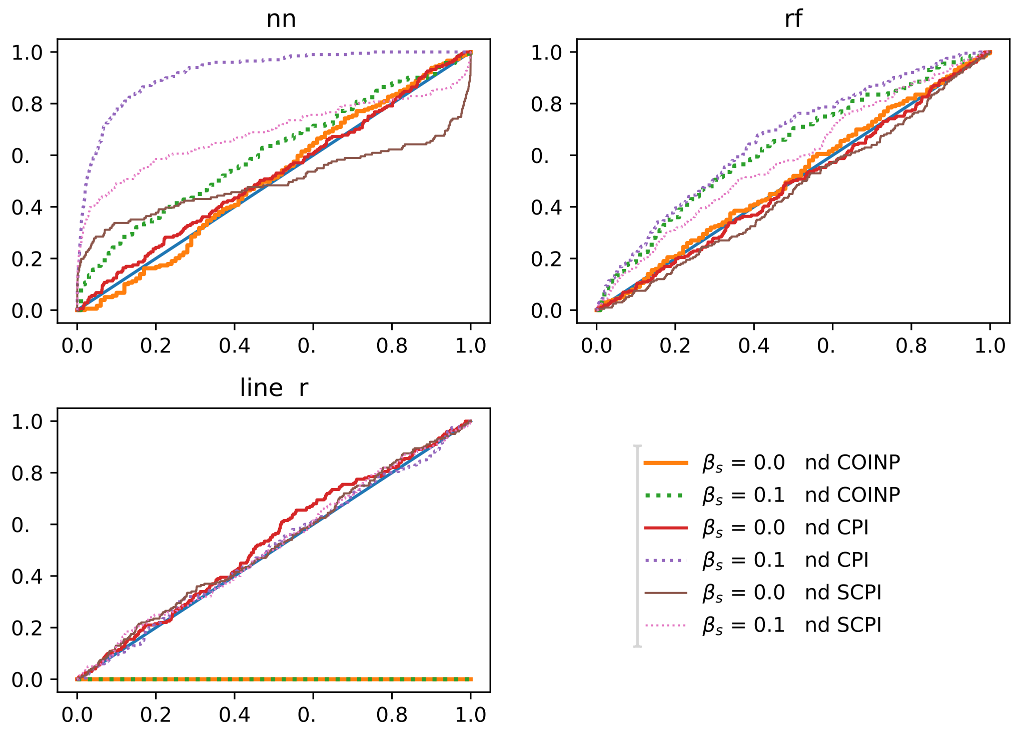


Figure 30 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI with $n = 10,000$ for a nonlinear distribution (distribution 6) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.1$ (alternative hypothesis).

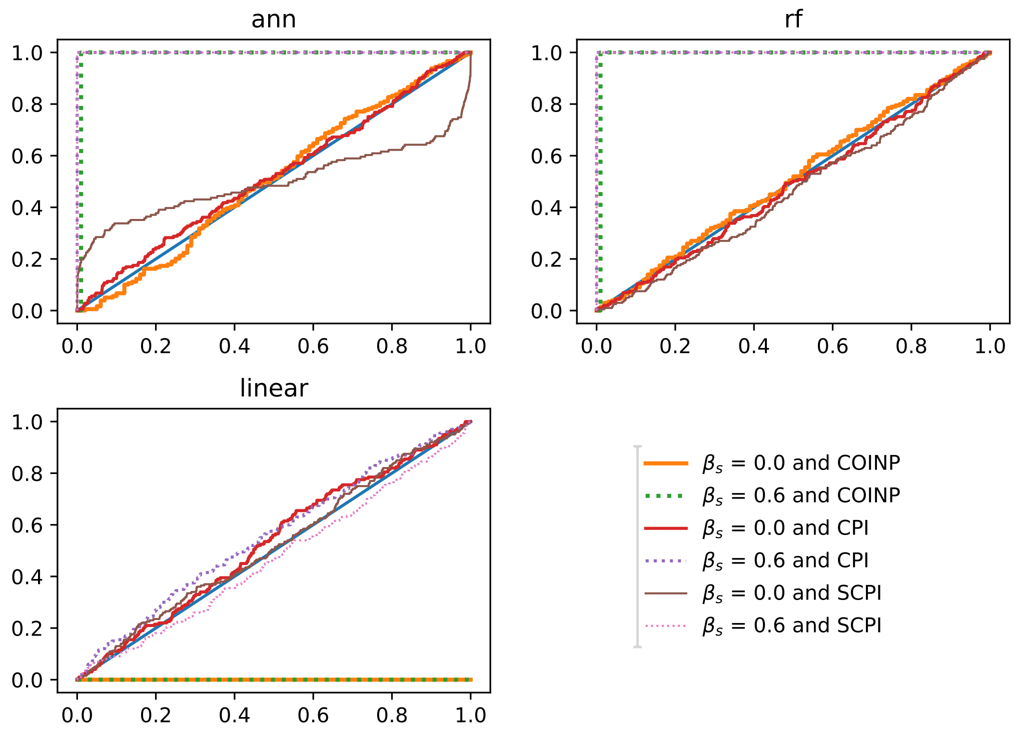


Figure 31 – Cumulative distribution of the p-values comparing COINP, CPI and SCPI with $n = 10,000$ for a nonlinear distribution (distribution 6) for $\beta_s = 0$ (null hypothesis) and for $\beta_s = 0.6$ (alternative hypothesis).

CHAPTER

6

CONDITIONAL DENSITY ESTIMATION

In this chapter, we present an introduction to the conditional density estimation problem and to Fourier Series. We also present our proposed method of conditional density estimation as well an study of the performance of such method using real world datasets with social, IT and astronomy related data.

6.1 Fourier series and density estimation

In this chapter we first give a brief introduction to Fourier series, and then show how to use it to estimate densities.

Let $L^2([0,1])$ be the linear space of continuous functions $f : [0,1] \to \mathbb{R}$ such that

$$\int_0^1 f(x)\mathrm{d}x \leq \infty$$

The usual inner product is defined by

$$\langle f, g \rangle = \int_0^1 f(x)g(x)\mathrm{d}x$$

This inner product induces the following norm and distance in $L^2([0,1])$:

$$\|f\| = \left(\int_0^1 f^2(x)\mathrm{d}x \right)^{1/2}$$

$$\sqrt{\mathbb{M}(f,g)} = \left(\int_0^1 (f(x) - g(x))^2\mathrm{d}y \right)^{1/2}$$

where $f, g \in L^2([0,1])$.

The sequence of functions $\{\phi_0, \phi_1, \phi_2, ...\}$ is called orthogonal system when

$$\langle \phi_i, \phi_j \rangle = 0$$

for $i \neq j$ and

$$\|\phi_i\| \neq 0$$

for all i. Furthermore, such a system is called orthonormal basis if for any $f \in L^2([0,1])$ there exists a unique sequence of scalars $\{\alpha_n\}_{n \in \mathbb{N}_+}$ such that

$$\left\| f - \sum_{k=1}^{I} \alpha_k \phi_k \right\| \to 0$$

as $I \to \infty$.

Also as of theorem 3.5.2 from Kreyszig (1989),

$$\alpha_k = \langle f, \phi_i \rangle$$

Thus, f has the following series representation:

$$\sum_{i=0}^{\infty} \langle f, \phi_i \rangle \phi_i$$

In this chapter we shall consider the Fourier basis where $\phi_i : [0,1] \to [-\sqrt{2}, \sqrt{2}]$ and

$$\phi_i(x) = \begin{cases} 1 & \text{if } i = 0 \\ \sqrt{2}\sin(\pi(i+1)x) & \text{if } i \in \{1,3,5,...\} \\ \sqrt{2}\cos(\pi i x) & \text{if } i \in \{2,4,6,...\} \end{cases}$$

Note that there are also many possible smoothness and/or boundary conditions (like twice differentiability) to ensure a somewhat fast convergence rate (see Efromovich (1999) for details). Intuitively, the less smooth the function f is, the larger will be the number of components needed in order to get a reasonable approximation. To give the reader some intuition on this, Figure 32 shows the curves for some of the components of the Fourier series. It can be seen that higher order components are needed in order to better "explain" less smooth functions.

In source code 4, we present a simple Python script that allows one to calculate the components of the Fourier series for arbitrary known functions.

Source code 4 – Obtaining Fourier series components of a function

```
from scipy import integrate
from scipy.stats import beta
from numpy import sin, cos, pi, sqrt

```

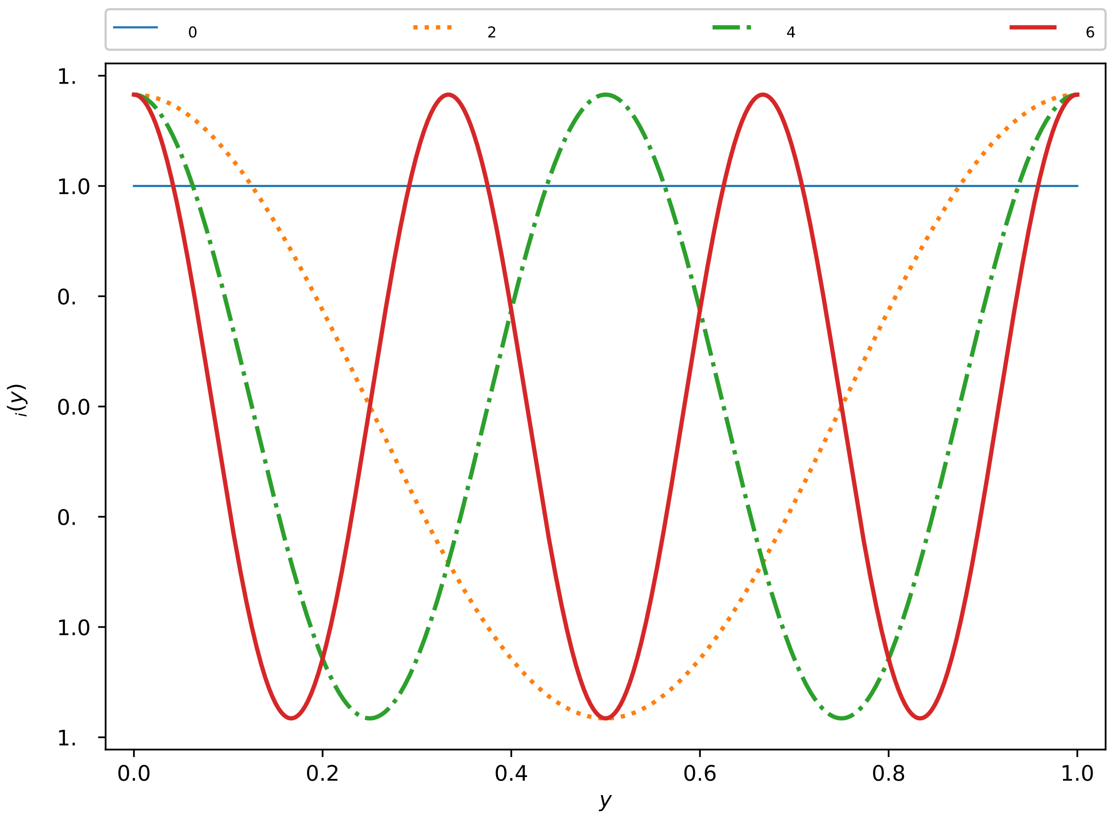


Figure 32 – Plot of some of the components of Fourier series: if a linear combination of such functions is used to approximate a function f, then it can be easily seen that the less smooth f is, the larger will be number of Fourier series components needed in order to better "explain" f.

```
#Function of interest
def func(t):
    return beta.pdf(t, 50.5, 10)

print("intercept", integrate.quad(func, 0, 1)[0])

for n in range(1, 10):
    component = lambda t: sqrt(2) * sin(n * 2 * pi * t)
    to_integrate = lambda t: func(t) * component(t)
    print("sin", n, integrate.quad(to_integrate, 0, 1)[0])

for n in range(1, 10):
    component = lambda t: sqrt(2) * cos(n * 2 * pi * t)
    to_integrate = lambda t: func(t) * component(t)
    print("cos", n, integrate.quad(to_integrate, 0, 1)[0])
```

The next two sections proceed with an exposition of the procedures for (unconditional) density estimation using frequentist and Bayesian approaches, respectively. While we don't directly use such estimator in this chapter, both of them will be useful in explaining our methodology of conditional density estimation as presented in chapter 6.2.

6.1.1 Frequentist Inference

Given i.i.d. random variables $Y_1, Y_2, ..., Y_n$ with density function $f : [0,1] \rightarrow \mathbb{R} \in L_2[0,1]$, a simple approach to infer f from a frequentist perspective using Fourier series is to use:

$$\hat{f}_I(y) = 1 + \sum_{i=1}^{I} \widehat{\alpha}_i \phi_i(y)$$

where

$$\widehat{\alpha}_i = \frac{1}{n} \sum_{j=1}^{n} \phi_i(Y_j) \approx \int \phi_i(y) f(y) dy = \langle \phi_i, f \rangle$$

This estimator is a special case of the modulator estimator which can found in Wasserman (2006), where we can also find the expected value and variance of each $\widehat{\alpha}_i$:

$$E(\widehat{\alpha}_i) = \int_0^1 \phi_i(x) f(x) \mathrm{d}x = \theta_i$$

$$Var(\widehat{\alpha}_i) = \frac{\int_0^1 \phi_i^2(x) f(x) \mathrm{d}x - \theta_i^2}{n}$$

as well as risk of the estimator $\hat{f}_I$:

$$R(\hat{f}_I; f) = E\left[\int_0^1 (\hat{f}_I(x) - f(x))^2 \mathrm{d}x\right] = \sum_{i=1}^{I} Var(\widehat{\alpha}_i) + \sum_{i=I+1}^{\infty} \theta_i^2$$

Therefore the choice of the estimator cutoff parameter I can be seen as bias-variance trade-off problem (in practice, a possible solution is to use cross-validation or data splitting to choose I).

Finally, we note that the estimate from $\hat{f}_I$ might not respect the constraint $\forall y \in [0,1], f(y) \geq 0$, in which case a "surgery" method is necessary (see Wasserman (2006) and Glad, Hjort and Ushakov (2003)).

6.1.2 Bayesian Inference

Just as in the previous section, given i.i.d. random variables $Y_1, Y_2, ..., Y_n$ with density function $f : [0,1] \rightarrow \mathbb{R} \in L_2[0,1]$, directly proceeding with the Bayesian inference (which is the focus of this chapter) of f using Fourier series is a somewhat difficult problem since we have to define and work with priors in the constrained space where $f(y) \geq 0$ for all $y \in [0,1]$.

One way to overcome this issue is to use the approach of sieve priors suggested by Scricciolo (2006), which places a prior directly on the coefficient vector β of the Fourier series expansion of $\log(f)$ (instead of f) so that conditionally on the threshold parameter (cutoff parameter) I we have:

$$f(y|I,\beta) = \frac{1}{g(\beta,I)} \exp\left\{\sum_{i=1}^{I} \beta_i \phi_i(y)\right\}$$

where g is a normalizing factor such that

$$g(\beta,I) = \int_0^1 \exp\left\{\sum_{i=1}^{I} \beta_i \phi_i(y)\right\} \mathrm{d}y$$

which is necessary in order to have $\int_0^1 f(y|I,\beta)\mathrm{d}y = 1$. Note that each β_i lives in $\mathbb{R}$, which solves the constrained space problem.

As a drawback, we introduced the difficulty of calculating a normalizing factor (using numerical integration) when evaluating the likelihood function. This could be handle with an MCMC sampler such as Stan Development Team (2014) which has built-in numerical integration capabilities and uses state-of-the-art sampling algorithms such as NUTS (HOFFMAN; GELMAN, 2014).

For more details on Bayesian density estimation using such prior, see Inácio, Izbicki and Salasar (2018), where one can find an extensive study in real and simulated datasets and also an application of such model to measure the distance between population densities.

6.2 Fourier series and conditional density estimation

We now consider the problem of conditional density estimation. Let $(X_1,Y_1),\ldots,(X_n,Y_n)$ be i.i.d. random vectors, where $Y_i \in \mathbb{R}$ is the response (label) and $X_i \in \mathbb{R}^d$ are covariates (features). Given that, problem of conditional density estimation can be stated simply as finding a good estimator $\hat{f}$ for the conditional density of $Y_k|X_k$, which we denote by $f(.|X_k) : [0,1] \to \mathbb{R} \in L_2[0,1]$, where $X_k = (X_{k1}, X_{k2}, ..., X_{kd})$. A simple solution for this problem is, for example, an ordinary least squares estimator:

$$\hat{f}(y|X_k) = \text{Gaussian}(X_k^T \hat{\beta}_{OLS}, \hat{\sigma}^2_{OLS})$$

Of course, such a simple estimator lacks flexibility for problems with complex structures both in terms of marginal density and in the structure of the covariates. Therefore, the goal of a good estimator is to be able to have considerable flexibility to model complex structure without incurring in excessive overfitting (bias-variance trade-off). In the next sections we review an already established method to deal with conditional density estimation using Fourier series, and introduce our proposed method which makes use of neural networks.

6.2.1 Flexcode

The Flexcode estimator (IZBICKI; LEE, 2017) is a natural extension of the frequentist density estimator method of section 6.1.1 to the conditional case.

It consists of two steps:

1. Estimate a regression function $r : \mathbb{R}^d \to [-\sqrt{2}, \sqrt{2}]^I$ where $r(.) = (r(.)_1, r(.)_2, ..., r(.)_I)$ and with $\phi_1(Y), \phi_2(Y), ..., \phi_I(Y)$ as targets and X as covariates. Such regression function can be obtained using a well known method such as OLS, Lasso and KNN.
2. Use the estimated regression to obtain following density estimate:

$$\widehat{f}(y_k|x_k) = 1 + \sum_{i=1}^{I} r(X)_i \phi(Y_k)$$

To understand why this procedure works, first notice that

$$E(f(y_k|x_k)) = 1 + \sum_{i=1}^{\infty} \left(\int_0^1 \phi_i(y) f(y|x_k) \mathrm{d}y \right) \phi(y_k) = 1 + \sum_{i=1}^{\infty} E(\phi_i(Y)|x_k)\phi(y_k)$$

and that the fitted value $r(X_k)_i$ of a regression of $\phi_i(Y)$ against X is itself an estimate of $E(\phi_i(Y)|X_k)$. It follows that the choice of a cut-point I is a problem of bias-variance trade-off (in similar fashion to the unconditional density estimator in 6.1.1) and, that in practice, this can be solved by cross-validation or data splitting.

6.2.2 Our proposed method: CDFSNet

Our approach for conditional density estimation builds on Flexcode in order to achieve better performance and scalability. In our initial tests, we directly applied the Flexcode strategy to neural networks. That is, we trained a Neural network M with:

- **Input**: a row vector input $x_k = (x_{k1}, x_{k2}, ..., x_{kd})$ of length d.
- **Output**: a row vector $(M(x_k)_1, M(x_k)_2, ..., M(x_k)_I)$ of length I.

where the estimated density given by

$$\widehat{f}(y_k|x_k) = 1 + \phi_1(y_k)M(x_k)_1 + ...\phi_I(y_k)M(x_k)_I$$

and the loss on the training set is given by

$$\sum_{i=1}^{n} \sum_{j=1}^{I} (\phi_j(y_i) - M(x_i)_j)^2$$

However, this bare bones Flexcode procedure has shown to perform poorly on neural networks, even after applying the various neural networks techniques to avoid overfitting and local minimum convergence that we described in section 2 and even after attempting two different "surgery" methods (in order to force the estimated densities to be positive and integrate to 1).

On the order hand, instead of calculating a squared error on the regression (as in step 1 of Flexcode), we can work directly with the loss function and we can also apply a exponential transformation in similar fashion to what is proposed in 6.1.1 (i.e.: calculate the Fourier components of $\log(f)$ instead of f). In this case, the performance increases dramatically. Therefore we have a Neural network N with:

- **Input**: a row vector input $x_k = (x_{k1}, x_{k2}, ..., x_{kd})$ of length d.
- **Output**: a row vector $(N(x_k)_1, N(x_k)_2, ..., N(x_k)_I)$ of length I.

where the estimated density given by

$$\widehat{f}(y_k|x_k) = \frac{\exp\{\sum_{i=1}^{I} \phi_i(y_k) N(x_k)_i\}}{g(N(x_k), I)}$$

and where $g(N(x_k, I))$ is a normalizing factor. Here, we work with the integrated squared distance between the true and estimated density functions as loss function:

$$\int_{\chi} \int_0^1 (f(y|x) - \widehat{f}(y|x))^2 \mathrm{d}y \mathrm{d}P(x).$$

This loss can be estimated by

$$n^{-1} \sum_{i=1}^{n} \int_0^1 (f(y|x_i) - \widehat{f}(y|x_i))^2 \mathrm{d}y = n^{-1} \sum_{i=1}^{n} \int_0^1 \left((\widehat{f}(y|x_i))^2 - 2 f(y|x_i) \widehat{f}(y|x_i) \right) \mathrm{d}y + k$$

where k is a constant. It follows then that the loss function on the training set is given by a numerical approximation to

$$n^{-1} \sum_{i=1}^{n} \left(\int_0^1 (\widehat{f}(y|x_i))^2 \mathrm{d}y - 2 \widehat{f}(y_i|x_i) \right) \tag{6.1}$$

where the integration can be estimated numerically using, for example, the trapezoidal rule.

Moreover, on preliminary tests we also considered the softplus transformation[1] which is defined by $\frac{1}{b} \log(1 + \exp(b * x))$ where $b = 1$ (PyTorch default[2]). Since softplus transformation lead to better performance than the exponential transformation for a fixed amount of Fourier Series components, we decided to use it instead. This implies that we are

[1] Softplus can also be used as an activation, but here we restrict its use as transformation to constrain the density function to positive values.

[2] We also tried other values for b, however the default value has shown to give better performance.

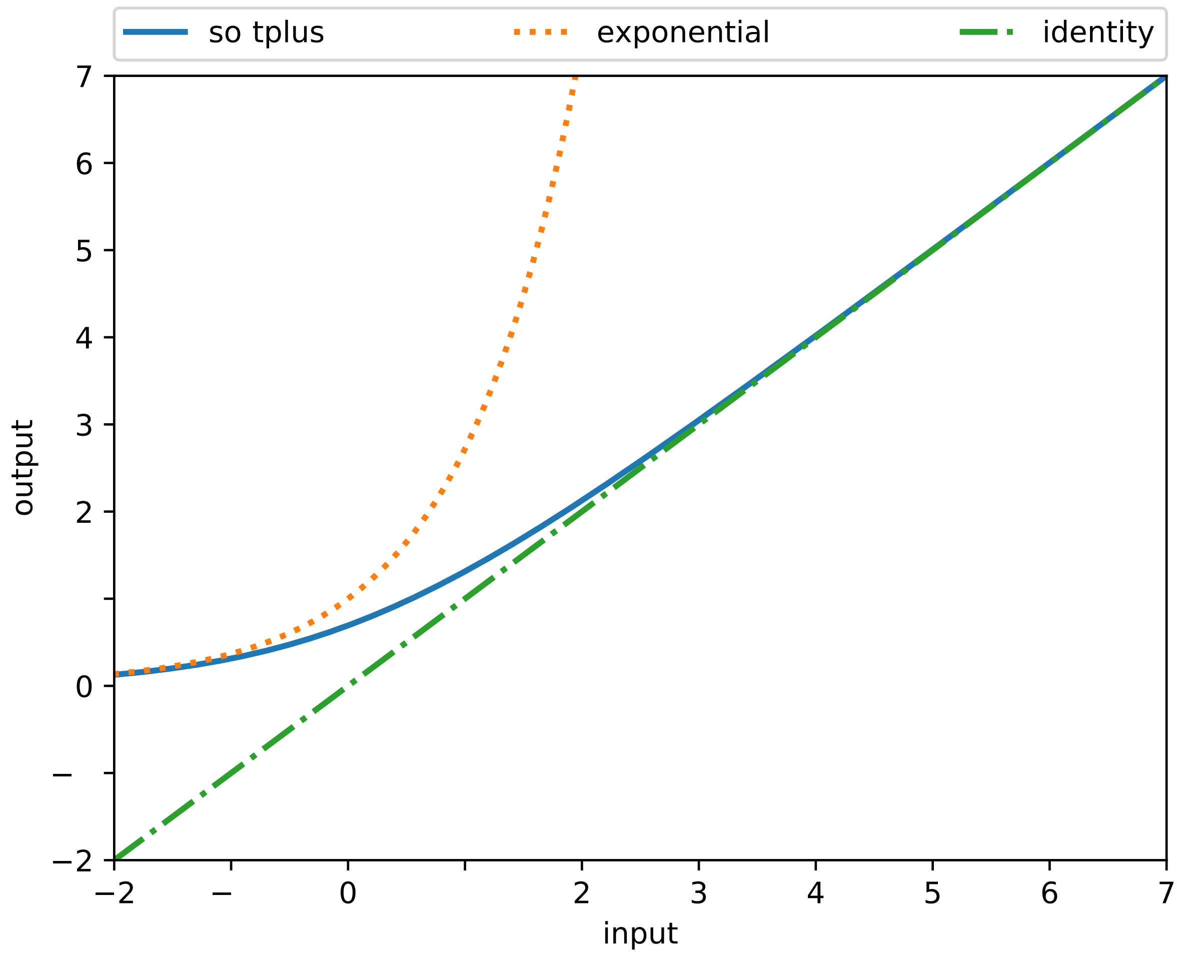


Figure 33 – Comparison of softplus, exponential and identity functions. Note that softplus function is always closer to the identity function that the exponential.

in fact calculating the Fourier basis components of softplus$^{-1}(f)$ and have the estimated density given by

$$\widehat{f}(y_k|x_k) = \frac{\text{softplus}\{\sum_{i=1}^{I} \phi_i(y_k) N(x_k)_i\}}{g'(N(x_k), I)}$$

where $g'(N(x_k), I)$ is another normalizing factor.

The intuition behind softplus giving some improvement over exponential is the fact that such transformation attempts to not significantly alter the value of its input (specially for large values), therefore potentially preserving the smoothness of the original (untransformed) density function. Figure 33 illustrates such property: softplus function is always to closer to the identity function that the exponential.

It is also worth noticing that this transformation and target loss function take advantage of the natural flexibility that neural networks have to minimize "arbitrary" loss functions and the speed GPUs can achieve when working with matrix multiplication which required for numerical integration inside the loss function. A Python package that implements the method that we therefore propose (and call CDFSNet) is available at <https://github.com/randommm/nncde>.

6.3 CDFSNet results and analysis

We now present a comparison of CDFSNet and three implementations of Flexcode using five real world datasets that we describe in the subsections. The Python source code of these analysis is available at <https://github.com/randommm/nncde_implementation>.

6.3.1 Datasets and preprocessing

We compare FlexCode with CDFSNet on the following datasets:

- **Spectroscopic dataset**: We take spectroscopic data from Izbicki and Lee (2016) and Izbicki, Lee and Freeman (2017). We take two subsets of the provided dataset. The first one with 10000 instances (in similar fashion to the one used by Izbicki and Lee (2016)) and the second one with 100000. We work with the redshift income as the response variable.
- **Pnad dataset**: We take a dataset from the Brazilian National Sample Survey of Households (PNAD), which is a research taken from Brazilian families and intends to extract information such as income, marriage, health, habitation and fecundity. For each attribute, we create an additional category to capture not available variables. We work with the family income as the response variable.
- **SGEMM dataset**: We take a dataset from Nugteren and Codreanu (2015) where the running time of a matrix-matrix product is measured, using a parameterizable SGEMM GPU kernel. For each combination of attributes, 4 runs were performed. For simplicity, we take the average of the 4 runs as the response variable.
- **Diamonds dataset**: We take the classical diamonds dataset which is readily available from ggplot2 library and Kaggle. We work with carat as the response variable.

We use the following preprocessing in our experiments:

- **Response variable preprocessing**: before training every model, we preproccess the response variable by taking its log and then transforming it to lie in the $(0, 1)$ interval. This is done for every dataset, with the exception of the spectroscopic dataset for which the response variable (the redshift of a galaxy) was already in the range of 0 to 1 in our received version.

- **Feature preprocessing**: before training the neural networks, we preproccess all the features to have to mean 0 and variance 1 for all datasets, with the exception of the SGEMM dataset where we use PCA-Whitening transformation.

The score evaluation being carried out in a test dataset[3] (which was not used to train the models nor in the validation procedure for early stopping the neural networks).

Table 17 – Score (greater is better) of different methods for a given dataset. Here we compared Flexcode (using nearest neighbors, XGBoosting and random forests) with CDFS-Net. NA represents a case where we were unable to train the model due to RAM limitations.

	Dataset				
	Spectroscopic		Pnad	SGEMM	Diamonds
Sample size	10000	100000	117939	241600	48940
Nº attributes	10	10	901	15	26
FC KNN	9.44	10.73	13.11	15.66	7.45
FC XGB	11.28	13.26	NA	18.86	15.56
FC RF	11.58	13.72	15.89	30.61	16.09
CDFSNet	13.57	16.63	15.95	54.74	19.79

6.3.2 Results

In Table 17, we present the score (the opposite of the integrated squared distance loss as given by equation 6.1) of CDFSNet using the Flexcode implementation of nearest neighbors, XGBoosting and random forests[4] as the comparison baseline and using the aforementioned datasets.

In Figure 34, where we present the estimated conditional probability density function of the Fourier ANN and Flexcode Random Forest methods for the SGEMM dataset (conditional on a point chosen at random).

6.3.3 Analysis

We notice from Table 17 that CDFSNet has outperformed all the other Flexcode based estimators. We note four possible reasons for such behaviour:

First, the neural network is trained to directly minimize the loss of interest, rather than several regression loss functions. Second, the Fourier series allows for a large number of Fourier series components to be used. Indeed, no hard cross-validation/data splitting

[3] The test dataset size was set to be the minimum between 5000 and 10% of the instances of the dataset.

[4] For such task, we used the Python Flexcode implementation available at <https://github.com/tpospisi/FlexCode> with the number of Fourier series components and some of the internal parameters of the estimators chosen by a data-splitting procedure.

procedure was necessary in order to choose the "cut point" of the Fourier series in our proposed method: a reasonably large neural network with 100 Fourier components works well "out-of-the-box" probably due to early stopping and dropout are already taking care of overfitting problems. On the other hand, for Flexcode estimators, a data splitting procedure generally dictates a much smaller number of Fourier series components due to the bias/variance tradeoff mentioned earlier.

Third, one of the limitations of the Flexcode method is that a Fourier series expansion might be negative in some regions, requiring some surgery procedures, and from Figure 34, we can see visually is that a large proportion of the density function is zeroed.

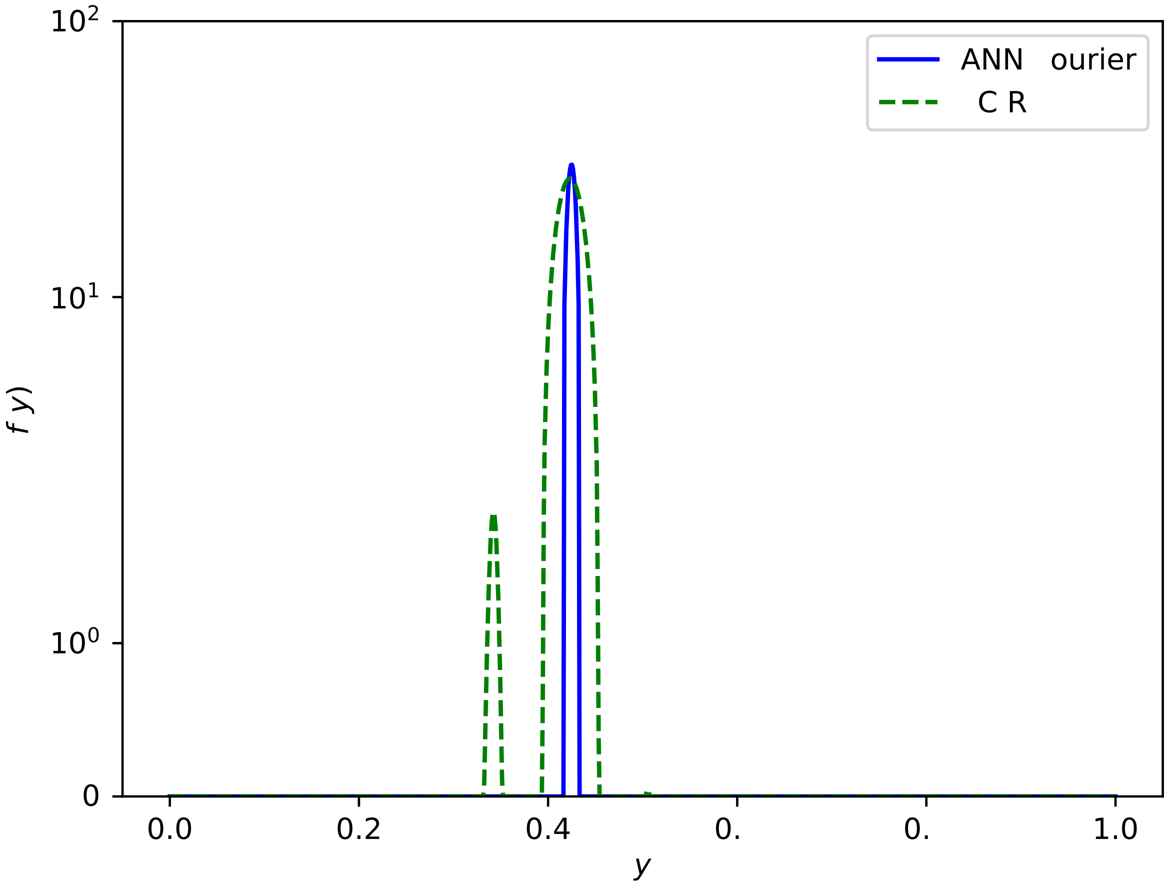


Figure 34 – Estimated probability conditional density function of the Fourier ANN and Flexcode Random Forest methods for the SGEMM dataset (conditional on a point chosen at random).

This also leads to a secondary effect of "stretching" the curve in points which already have positive density (in order force the density to integrate to 1). Intuitively, these effects may be causing an additional bias on the FlexCode density estimation for a given number of Fourier series components (a large number of Fourier series components might be able to overcome this issue, but at the price of larger variance). A theoretical study confirming this possibility is suggested as an extension of this chapter.

A fourth reason might be given by Zhang *et al.* (2016) which discusses the capabilities that neural networks have in achieving generalization without falling into overfitting possibly due to properties of stochastic gradient descendant.

CHAPTER 7 CONCLUSION

In this thesis, we have presented an introduction to dense neural networks, its properties, as well as the state-of-the-art methods of optimization and regularization.

We present an introduction to simulation studies and to the *sstudy* package, we have proposed a novel approach concerning the problem of conditional independence testing and have conclude that the proposed method yield consistent results. In future works, we plan to derive theoretical properties of such method and to extend it to sequential data.

Moreover, we have developed a novel framework for two sample comparison using variational autoencoders with the aim to bring the novel methods of machine learning to revisit and solve decades old classical problems of Statistics.

Finally, we have reviewed the concepts of Fourier series and conditional density estimation as well as an already established method of conditional density estimation using Fourier series and have proposed a novel method of conditional density estimation that combines both Fourier series and artificial neural networks and compared it to the well established one using five datasets.

BIBLIOGRAPHY

AKUZAWA, K.; IWASAWA, Y.; MATSUO, Y. Expressive speech synthesis via modeling expressions with variational autoencoder. **arXiv preprint arXiv:1804.02135**, 2018. Citation on page 55.

AN, J.; CHO, S. Variational autoencoder based anomaly detection using reconstruction probability. **Special Lecture on IE**, v. 2, n. 1, 2015. Citation on page 55.

AZZALINI, A.; VALLE, A. D. The multivariate skew-normal distribution. **Biometrika**, Oxford University Press, v. 83, n. 4, p. 715–726, 1996. Citation on page 78.

BERRETT, T. B.; WANG, Y.; BARBER, R. F.; SAMWORTH, R. J. **The conditional permutation test**. 2018. Citation on page 73.

BETANCOURT, M. The fundamental incompatibility of scalable hamiltonian monte carlo and naive data subsampling. In: BACH, F.; BLEI, D. (Ed.). **Proceedings of the 32nd International Conference on Machine Learning**. Lille, France: PMLR, 2015. (Proceedings of Machine Learning Research, v. 37), p. 533–540. Citation on page 55.

BIńKOWSKI, M.; SUTHERLAND, D. J.; ARBEL, M.; GRETTON, A. **Demystifying MMD GANs**. 2018. Citation on page 55.

BOTTOU, L. Stochastic gradient learning in neural networks. In: **In Proceedings of Neuro-Nîmes. EC2**. [S.l.: s.n.], 1991. Citation on page 31.

BREIMAN, L. Stacked regressions. **Machine Learning**, v. 24, p. 49–64, 1996. Citation on page 69.

______. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001. Citation on page 73.

BREIMAN, L.; CUTLER, A. **Random Forests for scientific discovery**. 2008. Available: <https://www.stat.berkeley.edu/~breiman/RandomForests/ENAR_files/frame.htm>. Citation on page 77.

BURDA, Y.; GROSSE, R.; SALAKHUTDINOV, R. Importance weighted autoencoders. **arXiv preprint arXiv:1509.00519**, 2015. Citation on page 56.

BURTON, A.; ALTMAN, D. G.; ROYSTON, P.; HOLDER, R. L. The design of simulation studies in medical statistics. **Statistics in Medicine**, Wiley, v. 25, n. 24, p. 4279–4292, 2006. Available: <https://doi.org/10.1002/sim.2673>. Citation on page 38.

CAMPOS, L. M. d. A scoring function for learning bayesian networks based on mutual information and conditional independence tests. **Journal of Machine Learning Research**, v. 7, n. Oct, p. 2149–2187, 2006. Citation on page 73.

CEREGATTI, R. d. C.; IZBICKI, R.; SALASAR, L. E. B. A nonparametric bayesian approach for the two-sample problem. In: POLPO, A.; STERN, J.; LOUZADA, F.; IZBICKI, R.; TAKADA, H. (Ed.). **Bayesian Inference and Maximum Entropy Methods in Science and Engineering**. Cham: Springer International Publishing, 2018. p. 231–241. ISBN 978-3-319-91143-4. Citations on pages 54 and 62.

CHALUPKA, K.; PERONA, P.; EBERHARDT, F. Fast conditional independence test for vector variables with large sample sizes. **arXiv preprint arXiv:1804.02747**, 2018. Citation on page 73.

COSCRATO, V.; ESTEVES, L. G.; IZBICKI, R.; STERN, R. B. **Interpretable hypothesis tests**. 2019. Citation on page 53.

COSCRATO, V.; INÁCIO, M. H. de A.; BOTARI, T.; IZBICKI, R. **NLS: an accurate and yet easy-to-interpret regression method**. 2019. Citation on page 48.

COSCRATO, V.; INÁCIO, M. H. de A.; IZBICKI, R. The NN-stacking: Feature weighted linear stacking through neural networks. **Neurocomputing**, Elsevier BV, Feb. 2020. Available: <https://doi.org/10.1016/j.neucom.2020.02.073>. Citation on page 69.

Dengsheng Zhang; Guojun Lu. Evaluation of similarity measurement for image retrieval. In: **International Conference on Neural Networks and Signal Processing, 2003. Proceedings of the 2003**. [S.l.: s.n.], 2003. v. 2, p. 928–931 Vol.2. Citation on page 55.

DEVROYE, L. Sample-based non-uniform random variate generation. In: ACM. **Proceedings of the 18th conference on Winter simulation**. [S.l.], 1986. p. 260–265. Citation on page 56.

DIKS, C.; PANCHENKO, V. A new statistic and practical guidelines for nonparametric granger causality testing. **Journal of Economic Dynamics and Control**, v. 30, n. 9, p. 1647 – 1669, 2006. ISSN 0165-1889. Computing in economics and finance. Available: <http://www.sciencedirect.com/science/article/pii/S016518890600056X>. Citation on page 73.

DJORK-ARNÉ, C.; UNTERTHINER, T.; HOCHREITER, S. Fast and accurate deep network learning by exponential linear units (elus). In: **Proceedings of the International Conference on Learning Representations (ICLR)**. [S.l.: s.n.], 2016. v. 6. Citations on pages 32, 48, 70, and 77.

DOERSCH, C. **Tutorial on Variational Autoencoders**. 2016. Citations on pages 56 and 58.

DORAN, G.; MUANDET, K.; ZHANG, K.; SCHöLKOPF, B. A permutation-based kernel conditional independence test. p. 132–141, 01 2014. Citation on page 73.

EFROMOVICH, S. **Nonparametric curve estimation: methods, theory and applications**. New York: Springer, 1999. ISBN 0-387-98740-1. Citation on page 92.

FAN, K.; WANG, Z.; BECK, J.; KWOK, J.; HELLER, K. A. Fast second order stochastic backpropagation for variational inference. In: **Advances in Neural Information Processing Systems**. [S.l.: s.n.], 2015. p. 1387–1395. Citation on page 56.

FISHER, A.; RUDIN, C.; DOMINICI, F. All models are wrong but many are useful: Variable importance for black-box, proprietary, or misspecified prediction models, using model class reliance. **arXiv preprint arXiv:1801.01489**, 2018. Citation on page 73.

FUKUSHIMA, K. Neocognitron. **Scholarpedia**, Scholarpedia, v. 2, n. 1, p. 1717, 2007. Available: <https://doi.org/10.4249/scholarpedia.1717>. Citation on page 35.

GLAD, I. K.; HJORT, N. L.; USHAKOV, N. G. Correction of density estimators that are not densities. **Scand J Stat**, Wiley-Blackwell, v. 30, n. 2, p. 415–427, jun 2003. Citation on page 94.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. v. 9, p. 249–256, 01 2010. Citations on pages 31, 70, and 77.

______. Understanding the difficulty of training deep feedforward neural networks. In: TEH, Y. W.; TITTERINGTON, M. (Ed.). **Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics**. Chia Laguna Resort, Sardinia, Italy: PMLR, 2010. (Proceedings of Machine Learning Research, v. 9), p. 249–256. Available: <http://proceedings.mlr.press/v9/glorot10a.html>. Citation on page 47.

GOOD, P. **Permutation tests: a practical guide to resampling methods for testing hypotheses**. [S.l.]: Springer Science & Business Media, 2013. Citation on page 73.

GRETTON A.AND BORGWARDT, K. M.; RASCH, M. J.; SCHÖLKOPF, B.; SMOLA, A. A kernel two-sample test. **Journal of Machine Learning Research**, v. 13, n. Mar, p. 723–773, 2012. Citations on pages 53 and 54.

HAHNLOSER, R. H. R.; SARPESHKAR, R.; MAHOWALD, M. A.; DOUGLAS, R. J.; SEUNG, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. **Nature**, Springer Science and Business Media LLC, v. 405, n. 6789, p. 947–951, Jun. 2000. Available: <https://doi.org/10.1038/35016072>. Citation on page 32.

HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. **The Elements of Statistical Learning**. New York, NY, USA: Springer New York Inc., 2001. (Springer Series in Statistics). Citation on page 29.

HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In: **2015 IEEE International Conference on Computer Vision (ICCV)**. IEEE, 2015. Available: <https://doi.org/10.1109/iccv.2015.123>. Citation on page 47.

HOFFMAN, M. D.; GELMAN, A. The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. **Journal of Machine Learning Research**, v. 15, p. 1593–1623, 2014. Citation on page 95.

HOLMES, C. C.; CARON, F.; GRIFFIN, J. E.; STEPHENS, D. A. Two-sample bayesian nonparametric hypothesis testing. **Bayesian Analysis**, International Society for Bayesian Analysis, v. 10, n. 2, p. 297–320, 2015. Citations on pages 53 and 54.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Networks**, Elsevier BV, v. 2, n. 5, p. 359–366, Jan. 1989. Available: <https://doi.org/10.1016/0893-6080(89)90020-8>. Citation on page 56.

HSU, W.-N.; ZHANG, Y.; GLASS, J. Learning latent representations for speech generation and transformation. **arXiv preprint arXiv:1704.04222**, 2017. Citation on page 55.

INÁCIO, M.; IZBICKI, R. **Conditional density estimation using Fourier series and neural networks**. 2018. Available: <https://bracis2018.mybluemix.net/files/anais-kdmile-2018.pdf>. Citation on page 28.

INÁCIO, M.; IZBICKI, R.; GYIRES-TÓTH, B. Distance assessment and analysis of high-dimensional samples using variational autoencoders. **Information Sciences**, Elsevier BV, Jul. 2020. Available: <https://doi.org/10.1016/j.ins.2020.06.065>. Citations on pages 28 and 44.

INÁCIO, M. H. A. **Simulation studies on Python using sstudy package with SQL databases as storage**. 2020. Citations on pages 27, 70, and 77.

INÁCIO, M. H. de A.; IZBICKI, R.; SALASAR, L. E. Comparing two populations using bayesian fourier series density estimation. **Communications in Statistics - Simulation and Computation**, Informa UK Limited, v. 49, n. 1, p. 261–282, Nov. 2018. Available: <https://doi.org/10.1080/03610918.2018.1484480>. Citations on pages 50, 54, 59, and 95.

INÁCIO, M. H. de A.; IZBICKI, R.; STERN, R. B. **Conditional independence testing: a predictive perspective**. 2019. Citations on pages 28, 44, and 55.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: BACH, F.; BLEI, D. (Ed.). **Proceedings of the 32nd International Conference on Machine Learning**. Lille, France: PMLR, 2015. (Proceedings of Machine Learning Research, v. 37), p. 448–456. Available: <http://proceedings.mlr.press/v37/ioffe15.html>. Citations on pages 33, 48, 70, and 77.

IZBICKI, R.; LEE, A. B. Nonparametric conditional density estimation in a high-dimensional regression setting. **Journal of Computational and Graphical Statistics**, Taylor & Francis, v. 25, n. 4, p. 1297–1316, 2016. Available: <https://doi.org/10.1080/10618600.2015.1094393>. Citation on page 99.

IZBICKI, R.; LEE, A. B. Converting high-dimensional regression to high-dimensional conditional density estimation. **Electron. J. Statist.**, The Institute of Mathematical Statistics and the Bernoulli Society, v. 11, n. 2, p. 2800–2831, 2017. Available: <https://doi.org/10.1214/17-EJS1302>. Citation on page 96.

IZBICKI, R.; LEE, A. B.; FREEMAN, P. E. Photo-z estimation: An example of nonparametric conditional density estimation under selection bias. **The Annals of Applied Statistics**, Institute of Mathematical Statistics, v. 11, n. 2, p. 698–724, 2017. Citation on page 99.

JENSEN, F. V. **An introduction to Bayesian networks**. [S.l.]: UCL press London, 1996. Citation on page 73.

KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. **CoRR**, abs/1412.6980, 2014. Citations on pages 31, 70, and 77.

KINGMA, D. P.; WELLING, M. Auto-encoding variational bayes. In: BENGIO, Y.; LECUN, Y. (Ed.). **2nd International Conference on Learning Representations, ICLR 2014**. [S.l.: s.n.], 2014. Citations on pages 27, 53, 55, 57, and 58.

KIRCHLER, M.; KHORASANI, S.; KLOFT, M.; LIPPERT, C. **Two-sample Testing Using Deep Learning**. 2020. Citation on page 54.

KLAMBAUER, G.; UNTERTHINER, T.; MAYR, A.; HOCHREITER, S. Self-normalizing neural networks. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 30**. Curran Associates, Inc., 2017. p. 971–980. Available: <http://papers.nips.cc/paper/6698-self-normalizing-neural-networks.pdf>. Citation on page 34.

KOLLER, D.; SAHAMI, M. Toward optimal feature selection. In: MORGAN KAUFMANN PUBLISHERS INC. **Proceedings of the Thirteenth International Conference on International Conference on Machine Learning**. [S.l.], 1996. p. 284–292. Citation on page 73.

KORNBLITH, S.; NOROUZI, M.; LEE, H.; HINTON, G. Similarity of neural network representations revisited. In: CHAUDHURI, K.; SALAKHUTDINOV, R. (Ed.). **Proceedings of the 36th International Conference on Machine Learning**. Long Beach, California, USA: PMLR, 2019. (Proceedings of Machine Learning Research, v. 97), p. 3519–3529. Citation on page 55.

KREYSZIG, E. **Introductory Functional Analysis with Applications**. [S.l.]: Wiley, 1989. ISBN 0471504599. Citation on page 92.

KRIZHEVSKY, A. **Learning multiple layers of features from tiny images**. [S.l.], 2009. Citation on page 60.

LARSEN, A. B. L.; SøNDERBY, S. K.; LAROCHELLE, H.; WINTHER, O. Autoencoding beyond pixels using a learned similarity metric. In: BALCAN, M. F.; WEINBERGER, K. Q. (Ed.). **Proceedings of The 33rd International Conference on Machine Learning**. New York, New York, USA: PMLR, 2016. (Proceedings of Machine Learning Research, v. 48), p. 1558–1566. Citation on page 55.

LI, Y.; MIN, M. R.; SHEN, D.; CARLSON, D.; CARIN, L. Video generation from text. In: **Thirty-Second AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2018. Citation on page 55.

LOPEZ-PAZ, D.; OQUAB, M. **Revisiting Classifier Two-Sample Tests for GAN Evaluation and Causal Discovery**. 2017. Citations on pages 54 and 69.

MANN, H. B.; WHITNEY, D. R. On a test of whether one of two random variables is stochastically larger than the other. **The annals of mathematical statistics**, JSTOR, p. 50–60, 1947. Citations on pages 45, 53, 54, and 64.

METCALFE, C.; THOMPSON, S. G. The importance of varying the event generation process in simulation studies of statistical methods for recurrent events. **Statistics in Medicine**, Wiley, v. 25, n. 1, p. 165–179, 2005. Available: <https://doi.org/10.1002/sim.2310>. Citation on page 38.

MONDAL, P. K.; BISWAS, M.; GHOSH, A. K. On high dimensional two-sample tests based on nearest neighbors. **Journal of Multivariate Analysis**, Elsevier, v. 141, p. 168–178, 2015. Citation on page 54.

MOONEY, C. Z. **Monte carlo simulation**. [S.l.]: Sage publications, 1997. Citation on page 38.

MORRIS, T. P.; WHITE, I. R.; CROWTHER, M. J. Using simulation studies to evaluate statistical methods. **Statistics in Medicine**, Wiley, v. 38, n. 11, p. 2074–2102, Jan. 2019. Available: <https://doi.org/10.1002/sim.8086>. Citation on page 38.

MUNDFORM, D. J.; SCHAFFER, J.; KIM, M.-J.; SHAW, D.; THONGTEERAPARP, A.; SUPAWAN, P. Number of replications required in monte carlo simulation studies: A synthesis of four studies. **Journal of Modern Applied Statistical Methods**, Wayne State University Library System, v. 10, n. 1, p. 19–28, May 2011. Available: <https://doi.org/10.22237/jmasm/1304222580>. Citation on page 38.

NUGTEREN, C.; CODREANU, V. Cltune: A generic auto-tuner for opencl kernels. In: **2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip**. [S.l.: s.n.], 2015. p. 195–202. Citation on page 99.

PARZEN, E. On estimation of a probability density function and mode. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 33, n. 3, p. 1065–1076, Sep. 1962. Available: <https://doi.org/10.1214/aoms/1177704472>. Citation on page 50.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KOPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. Pytorch: An imperative style, high-performance deep learning library. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; ALCHé-BUC, F. d'; FOX, E.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 32**. Curran Associates, Inc., 2019. p. 8024–8035. Available: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>. Citations on pages 48, 70, and 77.

PEARL, J. **Causality**. [S.l.]: Cambridge university press, 2009. Citation on page 73.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V.; VANDERPLAS, J.; PASSOS, A.; COURNAPEAU, D.; BRUCHER, M.; PERROT, M.; DUCHESNAY, E. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, v. 12, p. 2825–2830, 2011. Citation on page 77.

PFISTER, N.; BüHLMANN, P.; SCHöLKOPF, B.; PETERS, J. Kernel-based tests for joint independence. **Journal of the Royal Statistical Society. Series B: Statistical Methodology**, 03 2016. Citation on page 55.

RAMDAS, A.; TRILLOS, N. G.; CUTURI, M. On wasserstein two-sample testing and related families of nonparametric tests. **Entropy**, Multidisciplinary Digital Publishing Institute, v. 19, n. 2, p. 47, 2017. Citation on page 55.

REDDI, S. J.; KALE, S.; KUMAR, S. On the convergence of adam and beyond. In: **International Conference on Learning Representations**. [s.n.], 2018. Available: <https://openreview.net/forum?id=ryQu7f-RZ>. Citation on page 29.

ROSENBLATT, M. Remarks on some nonparametric estimates of a density function. **The Annals of Mathematical Statistics**, Institute of Mathematical Statistics, v. 27, n. 3, p. 832–837, Sep. 1956. Available: <https://doi.org/10.1214/aoms/1177728190>. Citation on page 50.

ROSSUM, G. van. **Python tutorial**. Amsterdam, 1995. Citation on page 37.

RUBIN, D. B. Bayesianly justifiable and relevant frequency calculations for the applies statistician. **The Annals of Statistics**, JSTOR, p. 1151–1172, 1984. Citation on page 50.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, Springer Science and Business Media LLC, v. 323, n. 6088, p. 533–536, Oct. 1986. Available: <https://doi.org/10.1038/323533a0>. Citation on page 35.

SCHAFFER, J. R.; KIM, M.-J. Number of replications required in control chart monte carlo simulation studies. **Communications in Statistics - Simulation and Computation**, Informa UK Limited, v. 36, n. 5, p. 1075–1087, Aug. 2007. Available: <https://doi.org/10.1080/03610910701539963>. Citation on page 38.

SCRICCIOLO, C. Convergence rates for Bayesian density estimation of infinite-dimensional exponential families. **The Annals of Statistics**, v. 34, n. 6, p. 2897–2920, 2006. Citation on page 95.

SEMENIUTA, S.; SEVERYN, A.; BARTH, E. A hybrid convolutional variational autoencoder for text generation. **arXiv preprint arXiv:1702.02390**, 2017. Citation on page 56.

SEN, R.; SURESH, A. T.; SHANMUGAM, K.; DIMAKIS, A. G.; SHAKKOTTAI, S. Model-powered conditional independence test. In: GUYON, I.; LUXBURG, U. V.; BENGIO, S.; WALLACH, H.; FERGUS, R.; VISHWANATHAN, S.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 30**. Curran Associates, Inc., 2017. p. 2951–2961. Available: <http://papers.nips.cc/paper/6888-model-powered-conditional-independence-test.pdf>. Citation on page 73.

SHAH, R. D.; PETERS, J. **The Hardness of Conditional Independence Testing and the Generalised Covariance Measure**. 2018. Citations on pages 28 and 73.

SILVERMAN, B. W. **Density Estimation/or Statistics and Data Analysis.** 1986. Citation on page 50.

SMIRNOV, N. Table for estimating the goodness of fit of empirical distributions. **The annals of mathematical statistics**, JSTOR, v. 19, n. 2, p. 279–281, 1948. Citations on pages 45, 53, 54, and 64.

SØNDERBY, C. K.; RAIKO, T.; MAALØE, L.; SØNDERBY, S. K.; WINTHER, O. Ladder variational autoencoders. In: **Advances in neural information processing systems**. [S.l.: s.n.], 2016. p. 3738–3746. Citation on page 56.

SORIANO, J. **Bayesian Methods for Two-Sample Comparison**. Phd Thesis (PhD Thesis) — Duke University, 2015. Citations on pages 53 and 54.

SPIRTES, P.; GLYMOUR, C. N.; SCHEINES, R.; HECKERMAN, D.; MEEK, C.; COOPER, G.; RICHARDSON, T. **Causation, prediction, and search**. [S.l.]: MIT press, 2000. Citation on page 73.

SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. **Journal of Machine Learning Research**, v. 15, n. 56, p. 1929–1958, 2014. Available: <http://jmlr.org/papers/v15/srivastava14a.html>. Citations on pages 33, 48, 70, and 77.

Stan Development Team. **Stan Modeling Language Users Guide and Reference Manual, Version 2.8.0**. [S.l.], 2014. Citation on page 95.

STROBL, C.; BOULESTEIX, A.-L.; KNEIB, T.; AUGUSTIN, T.; ZEILEIS, A. Conditional variable importance for random forests. **BMC bioinformatics**, BioMed Central, v. 9, n. 1, p. 307, 2008. Citation on page 73.

SUH, S.; CHAE, D. H.; KANG, H.-G.; CHOI, S. Echo-state conditional variational autoencoder for anomaly detection. In: IEEE. **2016 International Joint Conference on Neural Networks (IJCNN)**. [S.l.], 2016. p. 1015–1022. Citation on page 56.

TEICHER, H. *et al.* Identifiability of mixtures. **The annals of Mathematical statistics**, Institute of Mathematical Statistics, v. 32, n. 1, p. 244–248, 1961. Citation on page 57.

WALKER, J.; DOERSCH, C.; GUPTA, A.; HEBERT, M. An uncertain future: Forecasting from static images using variational autoencoders. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2016. p. 835–851. Citation on page 55.

WASSERMAN, L. **All of nonparametric statistics**. New York London: Springer, 2006. ISBN 0-387-25145-6. Citation on page 94.

WASSERSTEIN, R. L.; SCHIRM, A. L.; LAZAR, N. A. Moving to a world beyond ”p<0.05”. **The American Statistician**, Informa UK Limited, v. 73, n. sup1, p. 1–19, Mar. 2019. Available: <https://doi.org/10.1080/00031305.2019.1583913>. Citation on page 53.

WATSON, D. S.; WRIGHT, M. N. Testing conditional independence in supervised learning algorithms. **arXiv preprint arXiv:1901.09917v3**, 2019. Citations on pages 73 and 77.

______. Testing conditional predictive independence in supervised learning algorithms. **arXiv preprint arXiv:1901.09917v1**, 2019. Citation on page 76.

WECHSLER, S.; IZBICKI, R.; ESTEVES, L. G. A bayesian look at nonidentifiability: A simple example. **The American Statistician**, Taylor & Francis, v. 67, n. 2, p. 90–93, 2013. Citation on page 57.

WELCH, B. L. The generalization ofstudent's' problem when several different population variances are involved. **Biometrika**, JSTOR, v. 34, n. 1/2, p. 28–35, 1947. Citations on pages 45, 54, and 64.

WICKHAM, H. **ggplot2: Elegant Graphics for Data Analysis**. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. Available: <https://ggplot2.tidyverse.org>. Citation on page 81.

XU, B.; WANG, N.; CHEN, T.; LI, M. **Empirical Evaluation of Rectified Activations in Convolutional Network**. 2015. Citation on page 32.

YAN, X.; YANG, J.; SOHN, K.; LEE, H. Attribute2image: Conditional image generation from visual attributes. In: SPRINGER. **European Conference on Computer Vision**. [S.l.], 2016. p. 776–791. Citation on page 55.

ZHANG, C.; BENGIO, S.; HARDT, M.; RECHT, B.; VINYALS, O. **Understanding deep learning requires rethinking generalization**. 2016. Citation on page 102.

ZHOU, H.; ITHAPU, V. K.; RAVI, S. N.; SINGH, V.; WAHBA, G.; JOHNSON, S. C. Hypothesis testing in unsupervised domain adaptation with applications in alzheimer's disease. In: LEE, D. D.; SUGIYAMA, M.; LUXBURG, U. V.; GUYON, I.; GARNETT, R. (Ed.). **Advances in Neural Information Processing Systems 29**. [S.l.]: Curran Associates, Inc., 2016. p. 2496–2504. Citation on page 54.