uc3m | Universidad Carlos III de Madrid

Bachelor's Degree in Telecommunication Engineering

Academic Year 2018-2019

Bachelor Thesis

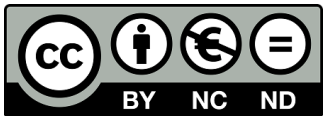
Entropy Triangles:

An application to measuring the transmission of entropy in Autoencoders

Adrian Manuel de Luis García

Tutor: Francisco José Valverde Albacete

Leganés, 10/23/2019



SUMMARY

The aim of this Undergraduate Thesis is to analyse the performance of Entropy Triangles when applied to typical Deep Neural Datasets.

Keywords:

DEDICATION

I am using the Dedication page as a place for a TODO list of things that apply everywhere or do not fit elsewhere. When taken care of, please remember to delete it.

General.

- I have tried to respect British English spelling conventions that you seem to be using. Please use en_GB as your default grammar in the LT box in the lower side of TeXStudio windows.
- Do not refer to this document as report but as Bachelor Thesis. If it becomes boring, use and abbreviation like B.Th. I've defined in the prelude_fva.tex a command BT doing exactly this.
- Check that variables are enclosed in math brackets. E.g. write X not X.
- Hay pares de palabras que no se pueden separar, como la palabra antes de una cita y la cita misma o Figure y el número de la figura. For that purpose you need a hard, non-separable space called in latex ~
- Figures and tables need to be attached to the first paragraph they are mentioned in and just below them, as a general rule.
- Explain all terms in equations either prior to it or after it. If quantities come from other equations, they do not need to be described again.

Section-by-section remarks:

- Introduction
  - Budget
- Theoretical results (fusion of chapters Basics and SoA)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AE	Autoencoder
DNN	Deep Neural Networks
ET	Entropy Triangles
GDPR	General Data Protection Regularization
IP	Information Bottleneck
ITL	Information Theorethic Learning
Knn	K Nearest Neighbors
ML	Machine Learning
MLP	Multilayer Perceptron
PCA	Principal Components Analysis

1. INTRODUCTION

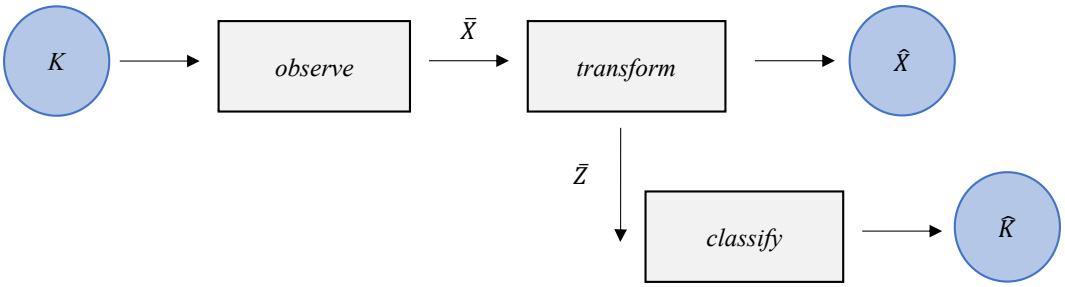
1.1. Background

Information has usually been referred to as one of the key aspects that every learning system needs in order to increase its capabilities and improve its performance. This is especially relevant for the Data Analytics field and the process of designing machine learning applications, where many experts usually refer to information as a powerful metric to indicate the success of their architecture at solving a particular problem. Due to the plethora of works already available online to master and use different techniques, the goal of this work is to asses the capabilities of these techniques and discuss its informational properties.

During the last few years, the topic of Big Data has become increasingly more important in our society. It is currently being used or developed in almost every industry in the world [1] and it is growing faster every day. While methods like Deep Neural Networks are beginning to be widely available, researchers tend to use the same approaches and steps to get their results. Rather than trying new methods, it is an industry standard to use a certain set of techniques when presented with a type of problem and then to try to optimize them. It is just assumed that these methods are powerful enough to find a suitable answer.

An conceptual diagram of this model can be found in Figure 1.1.a, where each block represents a different function inside an end-to-end informational model. Suppose we could only take measures outside each of the blocks and the contents or the methods being used inside are outside of the scope of the task. By not being able to access each one of the blocks to evaluate the processes inside them, we may only fix or change their inputs to try to improve our results.

(a) Conceptual representation of a classification process as a communication scheme.



(b) Our end-to-end scheme uses the information regarded from inside the transformation block and we then produce two outputs. First, the predicted (X) which we can use to measure our transformation accuracy. Secondly, the (Z), which contains compressed information about (X)

Fig. 1.1. Comparison between the classical view of a supervised classification in (a) versus the the model implemented for the purpose of this work in (b).

1.2. Objective

The model that I will be implementing will be an adaptation of the scheme on Figure 1.1.b. To further investigate on the boundaries of Information transmission , I will be using an **Autoencoder** to test the Information Bottleneck Principle [2].

In this report, I designed the following model which provides reliable information about the process of data compression and the limits of quantifiable information: +

- We have a random source K generating observations. Through a process of measurements, our system then will be provided with other random observations $\hat{X}$.
- The output observations are then fed to the Autoencoder, which will then reduce the $\overline{X}$ into another vector Z with a different length but retaining the information from the input vector in a compressed from.
- The Autoencoder also provides an output, which should be a reconstruction of the observations used to feed the Deep Learning structure.
- The $\overline{Z}$ is then used for the classifying task of choice to output the predicted labels.

Note that Figure 1.1b follows a similar model to that of Figure 1.1, but its transformation block is used to access the inside content of it rather than to provide a typical representation of an end-to-end transformation scheme. Although both of them typify a MIMO (Multiple Input Multiple Output) block, the Autoencoder is essentially an unsupervised transformation method, while the transformation block can be either composed by supervised or unsupervised task.

1.3. Outline

This report is divided into six Chapters, each one of them exploring different parts of the process of designing and testing the architecture proposed on Figure 1.1:

- **Chapter 1, Introduction** : Explains briefly the key concepts of the report as well as the cost of the project and the socioeconomic impact of Big Data.
- **??, Theoretical Background** : Includes basic information about the theory and methods used in the Bachelor Thesis as well as the state of the art.
- **Chapter 3, Development** : Key aspects to be aware of before starting the experimental phase of the project.
- **Equation 3.3.2, Result** : Presenting the results as well as the process to obtain them.
- **Chapter 4, Conclusion** : Final review of the project and the implications of it.

1.4. Socio-economic environment

1.4.1. Budget

The Budget for the project has been assigned according to three main costs : licenses, equipment and labor. The licenses used in this project are both included for Windows 10 and Mac OS X Mojave. On the other hand, the equipment includes the hardware and the resources needed for its adequate use. Finally, the labor costs are calculated using the metrics available at Indeed.com [3], and then considering that each programmer works 8 hours daily, 22 days monthly and 12 months yearly.

Licenses

PRODUCT	DESCRIPTION	COST
Windows 10	Pro Edition	50 €
MacOs	Mojave Edition	-. €
RStudio	1.2.1335	-. €
R	3.5	-. €
	Total:	50 €

Equipment & resources

PRODUCT	DESCRIPTION	COST
Laptop	Lenovo T480	1499 €
Laptop	Macbook Pro	1250 €
Internet Access	9 months	225 €
	Total:	2974 €

Labor

PRODUCT	DESCRIPTION	COST
Programmer	Hourly rate	8.5 €
	Total:	2550 €

TOTAL PROJECT COST: 5574 €

Table 1.1. TOTAL BUDGET COSTS TABLE

1.4.2. Socio-economic relevance

Big Data and Neural Networks have been referred to two of the major innovations of our decade. Many mainstream media articles have appeared in recent years discussing the impact that it will have in our lives [4]. Economic agents have been working on their development and how it will affect the way business are conducted in our modern era. In the same way, people have started to realize that data analytics and the management of information are important topics that affects each and everyone of us.

In this context, assessing the validity of results and their reliability becomes a must. From the medical field, where data can be critical to find the cure for a specific disease [5], to designing safety systems and creating better customer relationships for aviation compa-

nies [6], it is highly important for researchers to improve their methods to solve Big Data problems so they are able to find the correct solutions to these challenges. Moreover, thanks to the advances in hardware and information available to the public, everyone has access to a wide variety of information on the topic of Big Data. Taking all these points into account, establishing an empirical tool that is able to polish research and strengthen results in the data analytics field can have a positive impact that affects all stakeholders.

To sum up, Big Data has changed the way society is and by improving it we can be able to shape the technology and lives of tomorrow's people.

1.5. Regulatory framework

The regulatory framework of this project can be closely related with Regulation 2019/679 of the European Parliament, also known as General Data Protection Regulation [7]. The aim of this law is to protect the privacy of all of the individual citizens of the European Union (EU) and the European Economic Area. The GDPR gives the people of the EU the control of their personal data, which protects them from violations related to the use of their personal data in processes like Big Data or Data Analytics without their consent. It is also divided into eleven chapters, each one of them referring to different aspects of the legislation.

If data was to be used replicating the procedures presented in the Bachelor Thesis for commercial use (thus not applying Recital 18), some pieces of the regulation must be specially taken into consideration before doing so.

Fig. 1.2. View from the outside of the European Parliament.

Although all of the chapters introduced in the document are related to the topic of Big Data, as the possible holders of data we must take a deeper look into Chapter 3, which accounts for the rules that are to be applied to the data controller. A list of Articles that apply—and in which respect—follows:

- **Article 25** requires private data to be stored using the appropriate measures, and by further reading into it we see that Recital 78 mandates to store the data in such a way that it cannot be traced back to the source without additional information.
- At the same time, **Article 30** states that the processing activities related with such personal information must be recorded and kept.
- If a security break happened and private information from the users is compromised, **Article 33** mandates that the supervisory authority must be notified unless the breach is unlikely to affect individual rights.
- In case that the accessed information is likely to risk the rights and freedoms of a person, the individual must be notified (**Article 34**).
- To avoid breaches and help secure individuals' data,**Article 35** accounts for data protection impact assessments to be conducted if specific risks arise.

- **Article 37** requires a data protection officer to be designated in order to assist in the monitoring of the compliance with the Regulation and, in case of criminal conduct, measures have to taken by the regulatory agency (**Article 9** and **Article 10**).
- Organizations outside of the EU must also appoint a representative (**Article 27**).

This Bachelor Thesis complies with all of the Articles from the GDPR since no personal data was acquired or used for the purpose of our experiments. In case of performing the methods mentioned with the purpose of gaining financial benefit by using personal data, the previous guidelines should be followed to secure the privacy of the individual's information involved in the experiments.

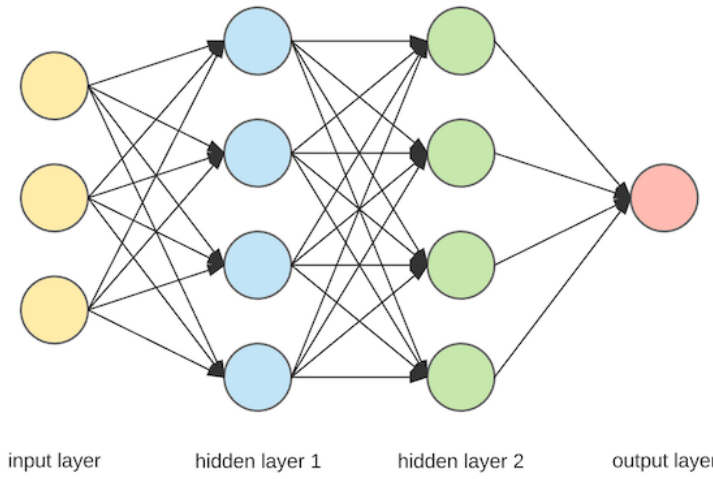
2. THEORETICAL BACKGROUND

In this chapter...

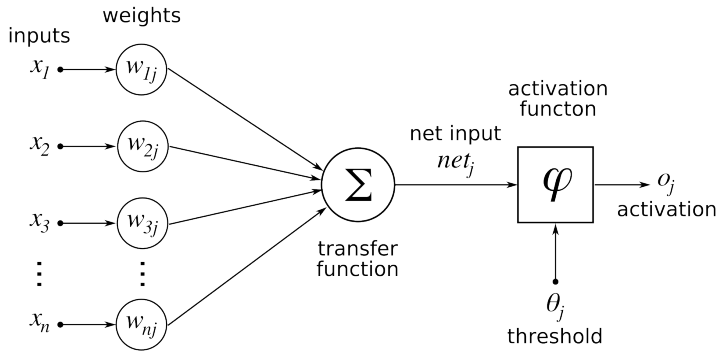
Include a paragraph explaining what you are doing in the chaper. Applicable to all chapters.

2.1. Deep Neural Networks

To clearly explain what an Autoencoder is, we must first lay the foundations of one of its core concepts: **Deep Neural Networks** (DNN). DNN constitute the basis of Deep Learning and they have proven themselves to be complex enough to tackle many of the data challenges of today.



(a) Basic depiction of a simple DNN composed by two hidden layers and a single 2-input/1-output system.



(b) Normal architecture of an artificial neuron.

Fig. 2.1. Basic depiction of a simple DNN composed by two hidden layers and a single 2-input/1-output system.

By analysing a, it can be noted that DNNs are comprised of multiple layers of units (or neurons) with a relatively small computing power which is calculated using the weights from previous layers combined with an activation function as seen on Figure 2.1. b. It also has an input (usually denoted as X) which is feed-forwarded to the hidden layers in the network and subsequently transformed into an output (commonly denoted as Y).

$$Output = f(bias_j + \sum_{i=0}^{n} w_i * x_i) \tag{2.1}$$

use the mechanism of references for all figures and subfigures. e.g like done in the paragraph above. The paragraph below has several problems with references of this type.

Neurons inside of the network can be modelled according to different types, although in this report we will be only taking into account two of them: the sigmoid $f(x) = \frac{1}{1-e-x}$

(Figure a) and the relu $f(x) = max(0, x)$ (Figure b) neurons. I chose to use this type of neurons because their activation functions are excellent for minimizing the reconstruction loss inside of the Autoencoder.

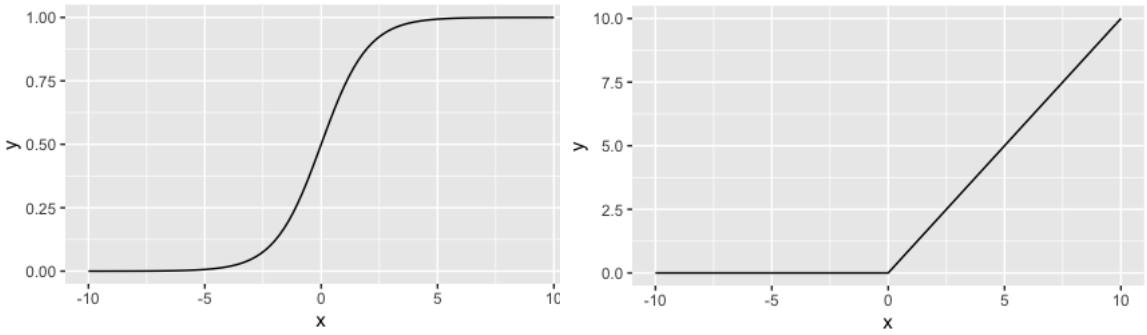


Fig. 2.2. The left graph represents a Sigmoid function, while on the right there is a Relu function

As seen on Figure 2.2, the Relu function has a relatively simple representation, which makes it less computationally expensive when compared to the Sigmoid. The Sigmoid function takes values between 0 and 1, which makes it specially useful when we want to place a classifier as the next task after the transformation.

Once a certain DNN structure has been activated, the artificial neurons inside of the network will start to propagate the information in the inputs in a non-linear manner in order to try to achieve a given established task, which in our particular case consists in transforming the representation of our input into a smaller but informationally more compressed form. Other experts can use the same principles to achieve other goals including decision-making, visualization, etc **this-needs-a-citation**.

2.2. Information Bottleneck Principle

All the notation below relates to random vectors, not to random variables. Accordingly, all variables have to be vectors such as $\overline{X}$.

The architecture of the Autoencoder is based upon the Information Bottleneck (IB), a method proposed in [2] whose principle relies on extracting the relevant information that an input random vector X contains about another output random variable Y. If we assume that there is some type of statistical dependency between X and Y, the relevant information can be defined as $I(X; Y)$.

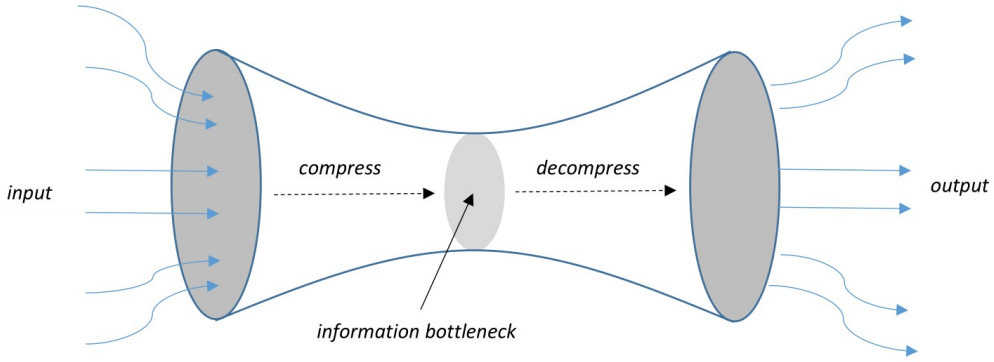


Fig. 2.3. Conceptual drawing of the Information Bottleneck.

If we want to obtain the optimal representation of X, we would want to capture the relevant information of X that contributes to an accurate prediction of Y. This term is known as the **minimal sufficient statistics citationToThis**, or simply just the mapping of X that retains $I(X;Y)$. The compressed version of X is denoted as T, and we can analyze the accuracy of our compression by comparing the prediction of Y that both T and X do. Essentially,our goal is to fulfill the following equation:

$$\min_{p(t|x)} I(X;T) - \beta \times I(T;Y) \tag{2.2}$$

In the following paragraph the indexing is of layers is not done apropriately. You need to use h_i and h_{i-1} throughout.

During the compressing process, and as depicted in Figure 2.3, the layers of a DNN only have access to the information that has been transferred to them via the previous layers of the system. This has a big effect on our network: the information not processed in the last immediate layer is essentially lost. This is the main reason why every layer should attempt to maximize $I(Y;hi)$ while trying to minimize $I(hi-1;hi)$, being hi a random layer situated inside of the coding or compressing layer and $hi-1$ its immediate predecessor. Here, it is important to consider then that we want to reduce the length of the layer to the minimum possible without affecting the predictive features of our model.

Each layer of our model should require shorter descriptions than the previous ones, being the first one with the longest description and the least compression. It must also be noted that every model will require a different set of layers and an architecture to fit its computational needs to the optimal level.

2.3. The Autoencoder

2.3.1. The Autoencoder Architecture

Applying the concepts introduced in both of the previous sections, the Autoencoder comes as a mixture of them. Figure 2.4 below presents it as a three part structure: A encoder, the middle layer and finally the decoder.

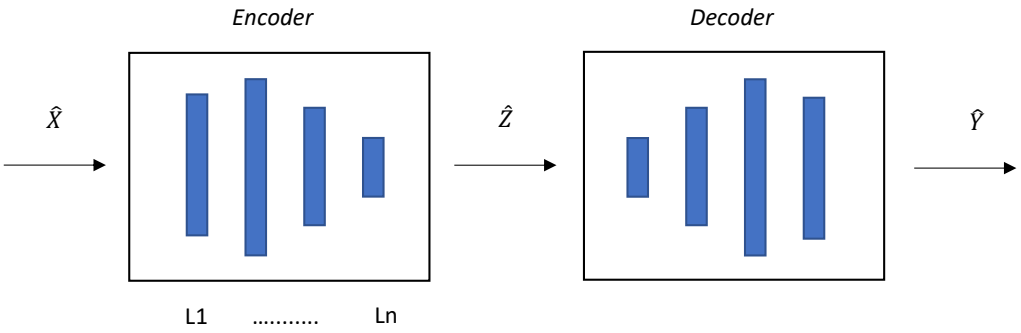


Fig. 2.4. Taking into account Figure 1.1b, this diagram depicts the inside of the transformation box. Both the Encoder and the Decoder have symmetric shapes, as well as an expansion layer—the second one in the encoder, the last-but-one one in the decoder— whose goal is to separate the qualities of our data as a sort of preparation for compression.

The theory of Autoencoders has been around since the 1980s **??**. As seen in Figure 2.4, the goal of the Autoencoder is not only to copy the input data and then transmit it through the network, but to retain the most important features of the data fed to it and to try to optimize the process of learning them. For the purpose of my research, I will only be considering the back-propagated Autoencoders, although it must be noted that other variants of them exist, such as the recirculated Autoencoders **citation**.

In our case, we will be also using an under-complete Autoencoder. For learning purposes, most of the times an Autoencoder is implemented it is usually more interesting to focus on its data compression capabilities rather than trying to copy the input onto the output. That is why we want to force the Autoencoder to provide us with a smaller dimension in the output on the encoder, which for the purpose of this report we will call $\overline{Z}$. The learning process tries to minimize the quantity:

$$L(x, g(f(x))) \tag{2.3}$$

L being a a loss function that penalizes $g(f(x))$ for being dissimilar from x.

One thing to take into account when designing an Autoencoder is that giving too much capacity to its layers can be counterproductive to the learning task. This means that when given too much capacity to work with they will tend to learn to avoid extracting information and rather to just copy the information, which is an undesirable outcome. On the other hand, trying to set an encoder to code the input signal into a single dimension could result in the loss of valuable information. Even with a very powerful decoder a very optimized Autoencoder will struggle to perform this task, specially when introducing very big sets data as the input.

Taking into account this issue, the general rule to design them is just by using trial and error. As we will see on Chapter 3.3.2, where I will be discussing the implementation of the Autoencoder, you can try different setups to reach the optimum middle layer size which accomplishes that there is no trade-off between augmenting its size and keeping its actual capabilities. The Tensorflow python tool provides us with the accuracy and loss variables that the Autoencoder is generating on each training epoch to check if your architecture is fulfilling your requirements. For the purpose of this B. Th., we will not only be taking into account those parameters, but we will also be using a tool to ensure that it transferring the higher quantity of information possible: the the Entropy Triangle.

2.3.2. Information Theorethic Learning on Autoencoders

The Information Theoretic Learning (ITL) concept was introduced in [8] to extend the Mean-Square error criterion to cost functions including information about the training data. The goal was to manipulate the information being carried in the data or, in other words, to find cost functions that would be able to directly influence information. The name ITL is a natural transition from the aforementioned process, as the machine learning function had to be independent from the informational transmission of data.

FVA: open up an “abbreviation section” and include ITL as abbrev.

The Autoencoder was first proposed to test the concepts of ITL on [9]. Using similar information-theoretic measures as the Entropy Triangle—such as the Mutual Information and entropy—the work of the researchers consists of trying to prove the validity of their theories by using the Autoencoder. Particularly, they are looking at the representation of

the $\hat{Z}$ and the reconstructed value of $\overline{X}$ to calculate the loss function.

$$cost = L(x, \overline{X}) + \lambda \times R(E, P) \tag{2.4}$$

In (2.4) L is the reconstruction cost function measuring the loss between the original X and the predicted $\overline{X}$ at the output of the Autoencoder. R is a functional regularization that uses information theoretic measures, P is the imposed prior and λ is a scale parameter that controls the strength of the regularization.

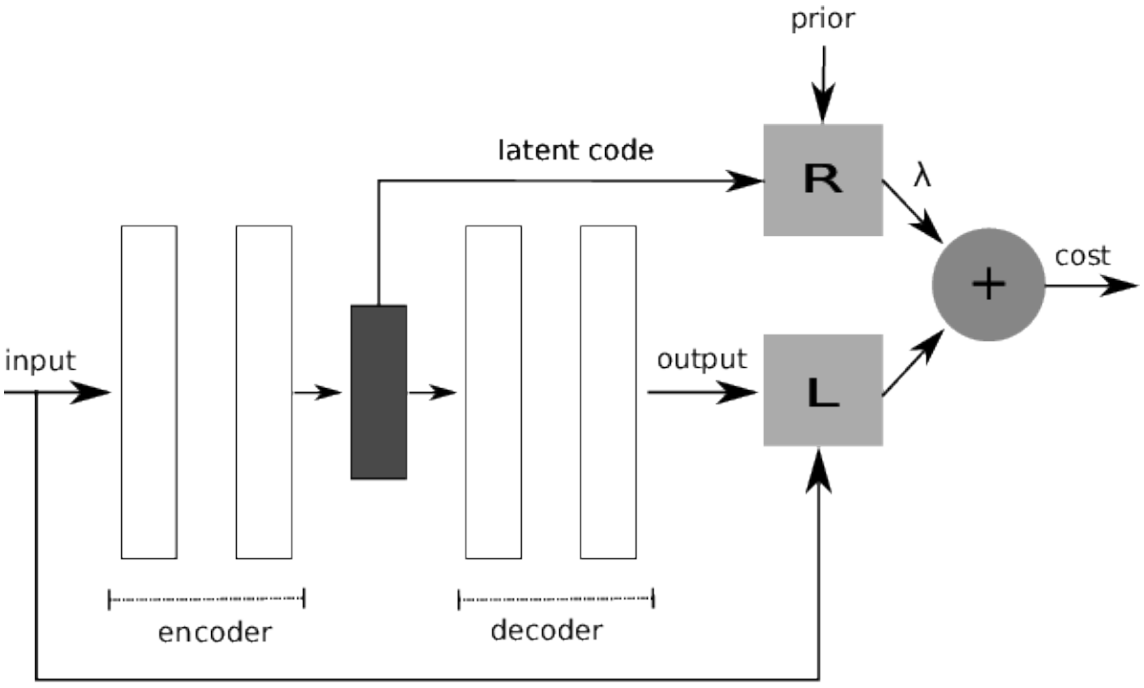


Fig. 2.5. Autoencoder illustrating Equation 2.4 from [9].

In [10] the authors expand on the use of Autoencoders and explore some of its key aspects regarding the bottleneck layer. By analysing the remarkable similarities between a transmission channel and the Autocoder—a structure also proposed in this report—they use widely available materials and datasets to evaluate the capacity of a stacked Autoencoder—as we do on our experiments—to demonstrate the ITL.

2.4. The Entropy Triangle

2.4.1. Motivation

In the previous section I have talked about how accuracy can be used to asses the validity of our model. In reality, the accuracy value can be somewhat misleading depending of the

task that we are implementing it to check if we are carrying out our data analysis correctly. This statement is specially true when talking about Classification [11].

The usage of data classification has greatly improved in the past years. Nowadays, there are multiple papers discussing how it can be used in a wide range of topics, from the medical field [12] to face recognition [13] . That is why it is specially important to correctly address the scope of your task and characteristics of your variables to be able the get the maximum performance out of your predictions.

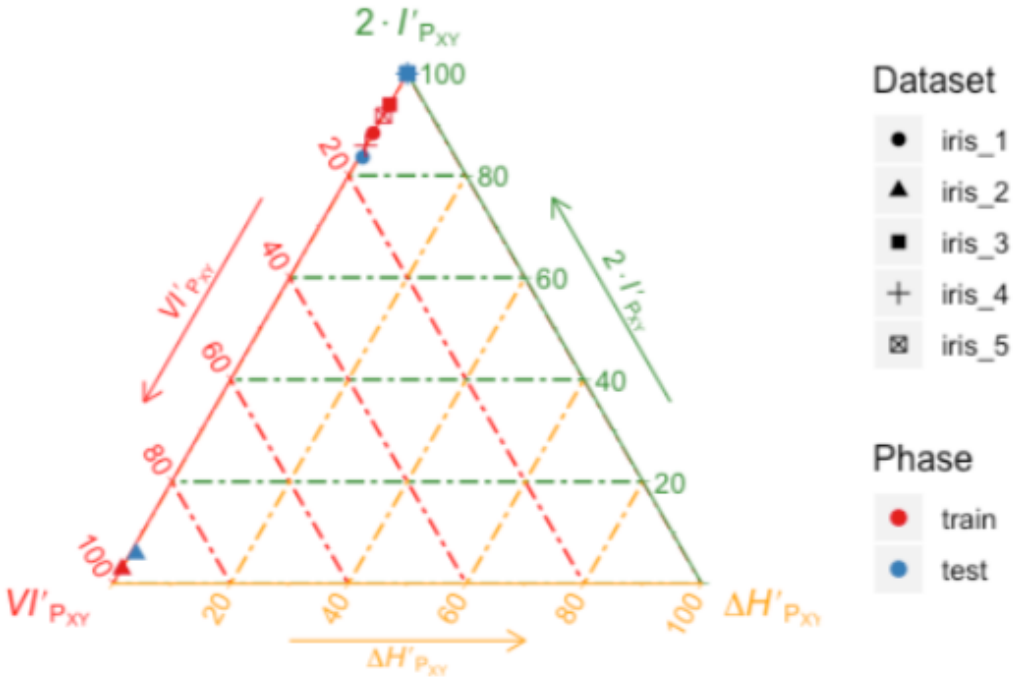


Fig. 2.6. Example of use of a typical Entropy Triangle. In this case, a Knn classification method is being tested in the triangle.

The real question comes when you have your results and you try to understand them. In [11] a very important reality is shown: the accuracy performance criterion is a very intuitive but misleading tool. Another highly important concept is introduced here: the accuracy paradox.

The accuracy paradox shows some of the defects that classifiers have been presenting for a long time now . For example, in some cases predictive classifiers which have given a lower accuracy power have also proven to display a higher predictive accuracy than others

with a higher accuracy. The reason for this most of the times is that we have trained our classifier with a single class that contains the majority of the data. In those cases, our classifier will just assign all of the input values to belong to the majority class since that is where the greater probability lies upon. This one is a very easily identifiable case of an *imbalanced (training) data set.* Moreover, since most of the times our data is gathered in controlled conditions, we risk the problem of this issue showing up more than we would wish for.

Once it is realized that the methods previously employed can lead to misleading results, a new realization arises: there is a need for a better measure of classification that takes this issue into consideration. As seen in [11], the Entropy Triangle and its features show some numeric example that fit the requirements and scope of our task, so I will be using it. Bur firstly, I will explain the basics of its functionality.

2.4.2. Architecture

The Entropy Triangle was introduced in [14] as a way of solving the problems presented in the previous sections. We consider the transmission of information through a channel as two random variables, named X for the input and Y for the output. Note that on Figure 1.1 we used a different naming standard, K and $\hat{K}$, but the new reference names are now implemented for the sake of easier computations. In Figure 2.7 we can see a classical information-diagram which simply shows the entropy relationship between X and Y or P_{XY}. From that Figure we can also asses some equations from it: The Mutual Information, which quantifies the stochastic force between P_X and P_Y appears twice in the diagram both as:

$$MI_{P_{XY}} = H_{P_X * P_Y} - H_{P_{XY}} \tag{2.5}$$

and as

$$MI_{P_{XY}} = H_{P_X} - H_{P_{X|Y}} \tag{2.6}$$

The Variation of information are embodied by the sum of the two red areas and represents the residual entropy, which is not used in the binding of the variables:

$$VI_{P_{XY}} = H_{P_{Y|X}} + H_{P_{X|Y}} \tag{2.7}$$

And both equations mentioned before together with $\Delta H_{P_{X*P_Y}}$, which represents the divergence between the joint distribution where P_X and P_Y are independent and the uniform distributions with the same cardinality of events as the previously mentioned P_X and P_Y, form:

$$H_{U_{X*U_Y}} = \Delta H_{P_{X*P_Y}} + 2 * MI_{P_{XY}} + VI_{P_{XY}} \tag{2.8}$$

In which case $H_{U_{X*U_Y}}$ is the outer rectangle with both the uniform distributions of the input and the output.

Once we have obtained equation2.8 we will normalize it using the $H_{U_{X*U_Y}}$ and thus forcing the variables involved in our calculations to be bounded by 0 and 1, as it can be seen on:

$$1 = \Delta' H_{P_{X*P_Y}} + 2 * MI'_{P_{XY}} + VI'_{P_{XY}} \tag{2.9}$$

Which also will wield the following equation:

$$0 \leq \Delta' H_{P_{X*P_Y}}, MI'_{P_{XY}}, VI'_{P_{XY}} \leq 1 \tag{2.10}$$

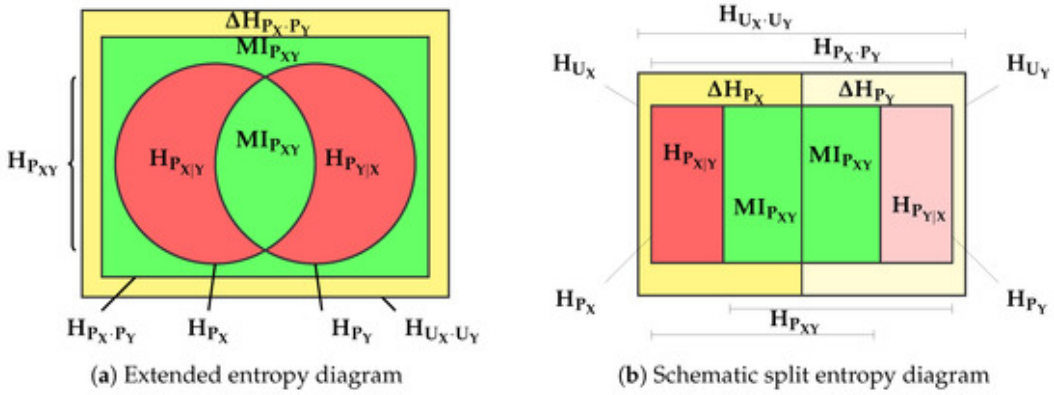


Fig. 2.7. Diagram representing an ET applied to a bi-variate distribution from [15].

By applying both equations 2.9 and 2.10 we will get a point in the normalized space $\Delta' H_{P_{X*P_Y}} x 2 * MI'_{P_{XY}} x VI'_{P_{XY}}$. Each P_{XY} can be characterized as $F(P_{XY}) = [\Delta' H_{P_{X*P_Y}}, 2 * MI'_{P_{XY}}, VI'_{P_{XY}}]$. The resulting diagram will be an equilateral triangle, where the coordinates are $F(P_{XY})$ and every bi-variate distribution is shown as a point in the diagram. Every zone of the triangle has certain characteristics related to it.

We can also divide equation 2.8 to obtain new representations of the split balance equations with respect to X and Y,

$$H_{U_X} = \Delta H_{P_X} + MI_{P_{XY}} + H_{P_{X|Y}} \rightarrow 1 = \Delta H'_{P_X} + MI'_{P_{XY}} + H'_{P_{X|Y}} \tag{2.11}$$

and,

$$H_{U_Y} = \Delta H_{P_Y} + MI_{P_{XY}} + H_{P_{Y|X}} \rightarrow 1 = \Delta H'_{P_Y} + MI'_{P_{XY}} + H'_{P_{Y|X}} \tag{2.12}$$

Both equations are normalized by using both H_{U_X} and H_{U_Y} respectively. Now, we have new representations for X and Y in the 2-simplex triangle created before. The representation seems to split in two and have $F(P_X) = [\Delta' H_{P_X}, MI'_{P_{XY}}, H'_{P_{X|Y}}]$ and $F(P_Y) = [\Delta' H_{P_Y}, MI'_{P_{XY}}, H'_{P_{Y|X}}]$ coordinates.

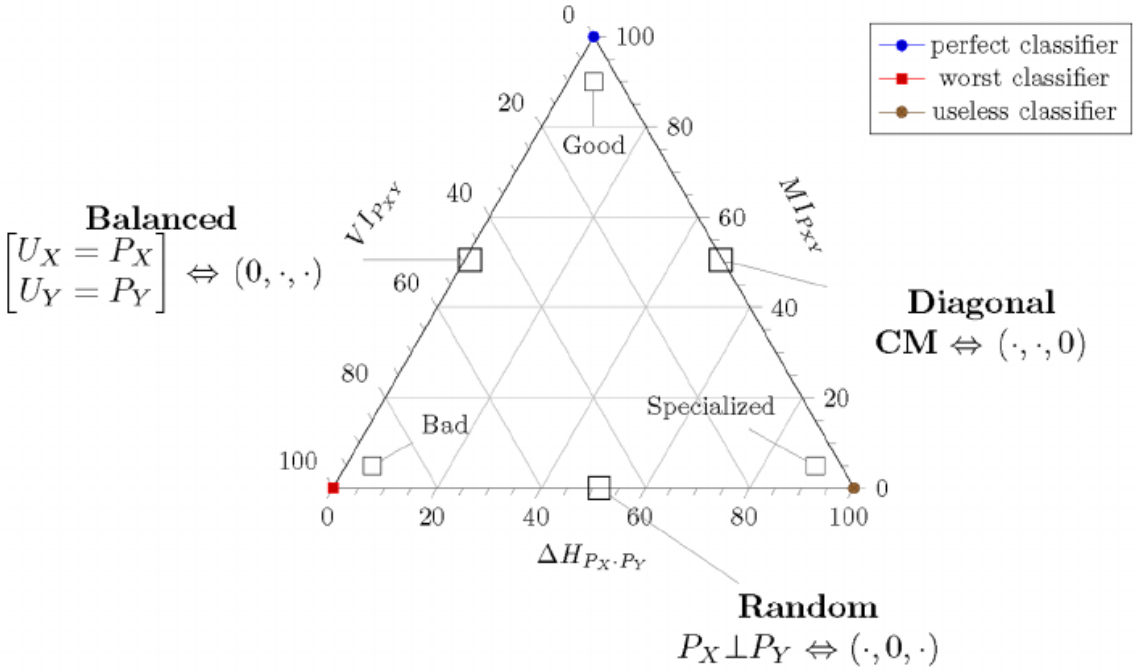


Fig. 2.8. From [15] Schematic of an Entropy Triangle. The labels are placed close to the side of the triangle that is related to the feature mentioned in it.

Most of the equations included here can be found in [15] which discusses most of the points mentioned here along with some examples of use and much more information on the tool and different implications of its usage.

The triangle itself works as a great tool to characterize the performance of your model. The position of the coordinates assesses the quality of your classifier. It can be appreciated from Figure 2.8 that classifiers which are at the apex or close to it obtain the highest

accuracy possible on balanced datasets and transmit a lot of mutual information, which makes them the best classifiers possible. On the other hand, when the coordinates of the classifier are very close to the left apex, we will be working with balanced data but the classifier will be doing a very bad work with it, we are dealing with the worst classifier. Finally, those who are located at the right apex or close to it represent the accuracy paradox mentioned before, which a highly specialized classifier which tries to work with very unbalanced data.

One example of use of the Entropy Triangle would be to use the true labels K and the predicted labels $\hat{K}$ to generate a confusion matrix. Using an algorithm, the entropy of the process is calculated according the values in the confusion matrix, which will then provide us with a coordinate inside of the ET. It will provide visual data to assess the feasibility of the task and its effectiveness.

SO FAR 22/09/2019

3. EXPERIMENTAL RESULTS

In this chapter we motivate the datasets and classifiers we use to test our assumptions.

3.1. Dataset Selection and Exploratory Analysis

Before diving into the application of the Autocoder, we first have to take a look into the data that we will be using on our project. We want to use a wide variety of them to test our tool in order to find out if the assessments made on the previous section hold when applied on some of the most commonly datasets used in the industry. When selecting between the huge amounts of information available online, it was our interest to prioritize them according to a series of qualities. They needed to:

- Have different instance cardinalities, ranging from easily handable datasets (Iris) to more complex and computational harder (MNIST).
- Different classification task types, from binary to categorical.
- Clearly distinct balance in the datasets, which will help us understand the potential of our tool.

The following section will help understand these points and its implications for our task. To do so, a brief description of them and their role on achieving our desired results will be stated. Descriptions of each dataset will include statistics on their distributions as well as why they were chosen as a viable candidate for our goal. Mainly we will be relying on simple computations, such as the median, and trying to plot histograms or similar figures to have visible and more user friendly inputs to understand why some steps will be applied on future scripts.

This task is necessary if we want to have a hint of the outcome of the processes that will take place on the datasets, as well as to shed light onto the difficulty to adapt our scripts and architectures when moving from one dataset to another one. It also provides some information about why they were chosen as viable candidates in our experiments.

3.1.1. Anderson's Iris

(Anderson's) Iris is a multivariate dataset published in [16]. The data itself consists of 50 samples from each of three species of Iris: *Iris setosa*, *I. virginica* and *I. versicolor*. Each exemplar had four of their characteristics recorded: the length and the width of the sepals and petals in centimeters.

Based on the distribution of the measurements obtained from the dataset, Fisher's data has been referred to as one of the basic staples of data classification due to the fact that it is divided into two clearly differentiable clusters, one containing Iris setosa and the other one including both virginica and versicolor. Without the labels provided by Fisher, the classifying task becomes more complex. This feature is specially useful to highlight the differences between unsupervised and supervised classification tasks.

Variable name	**Sepal Length**	**Sepal Width**	**Petal Length**	**Petal Width**
Minimun	4.30	2.00	1.00	0.10
Median	5.80	3.00	4.350	1.30
Mean	5.84	3.06	3.76	1.20
Maximun	7.90	4.40	6.90	2.50

Table 3.1. R'S SUMMARY METHOD ON IRIS.

Table 3.1 contains summaries of the features of iris, and by looking at them we can already see that the data is very balanced on every feature as the distance between the minimum and the maximum divided by two is approximately the Mean, which also has a close value to the Median. To us, this means that the data is centred over a certain value, which will allow us to simplify our tasks by doing some type of translation transformation over the data before fitting it through our tools. Regardless, plotting will also help us discern the real difficulty of our classification task.

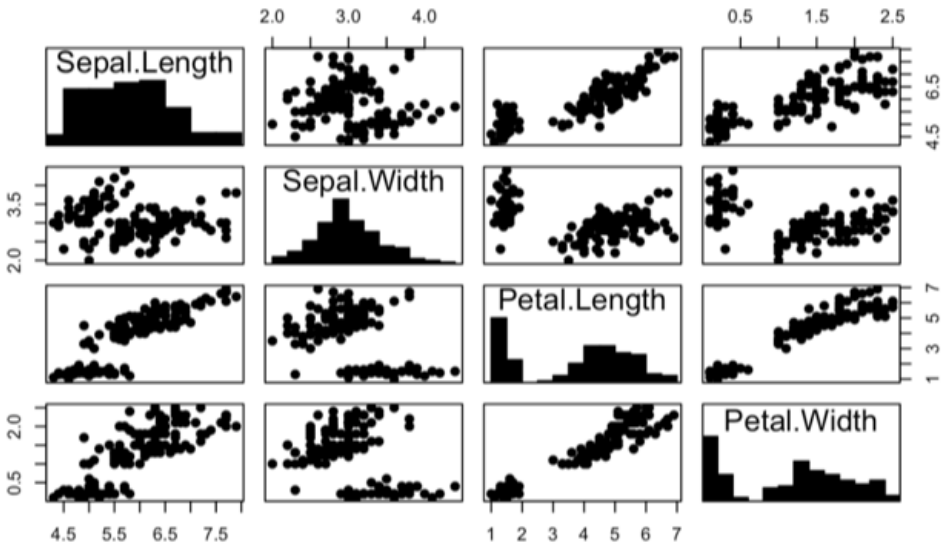


Fig. 3.1. Using the pairs function summary

Taking a look at Figure 3.1, The histogram in the middle is showing some higher values in different ranges on our data. If we hadn't observed the summary values in Table 3.1 , we could have struggled with coming up with a general idea behind the dataset. But sometimes histograms are a little bit more troubling to read into, so plotting Figure 3.2 can help understand the relationships between classes and their features. For example, the iris dataset would be a good fit for a knn classifier, as its data classes and observations are clearly distinguishable when using Petal Width and Petal Length rather than when using Sepal Width and Sepal Length.

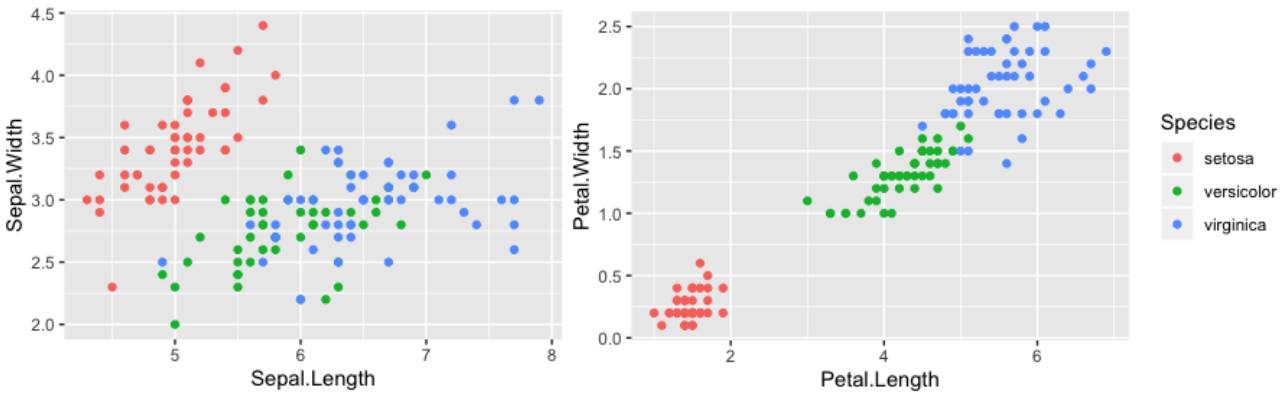


Fig. 3.2. Plotting of two different features of Iris with respect to the class label. As it can be seen here, depending of how you organise your data you can get more efficient classifiers and clusters.

3.1.2. MNIST

Find citation for MNIST.

The MNIST dataset **MNIST** is a large database of handwritten digits widely used in Data Science for experimenting. It is mostly used for image processing and it is composed by samples of handwriting taken from the National Institute of Standards and Technology original datasets. It was later normalized to fit into 28x28 pixel images. Every image has vector X of values $24 \times 24 = 784$ between 0 and 255 ,which defines the shape and shades of grey of the number in the image. Then, the Y class contains vectors with a character which can be related to its corresponding pixelized image by a classification method.

The database contains 60 000 training images and 10 000 testing images for X and Y. Many different authors [17], have tried to achieve the lowest error rate on it. Convolutional neural networks managed to get a very low error of around 0.23, although other methods such as support-vector machine get an error of as low as 0.56 too[1].

[1] 18.

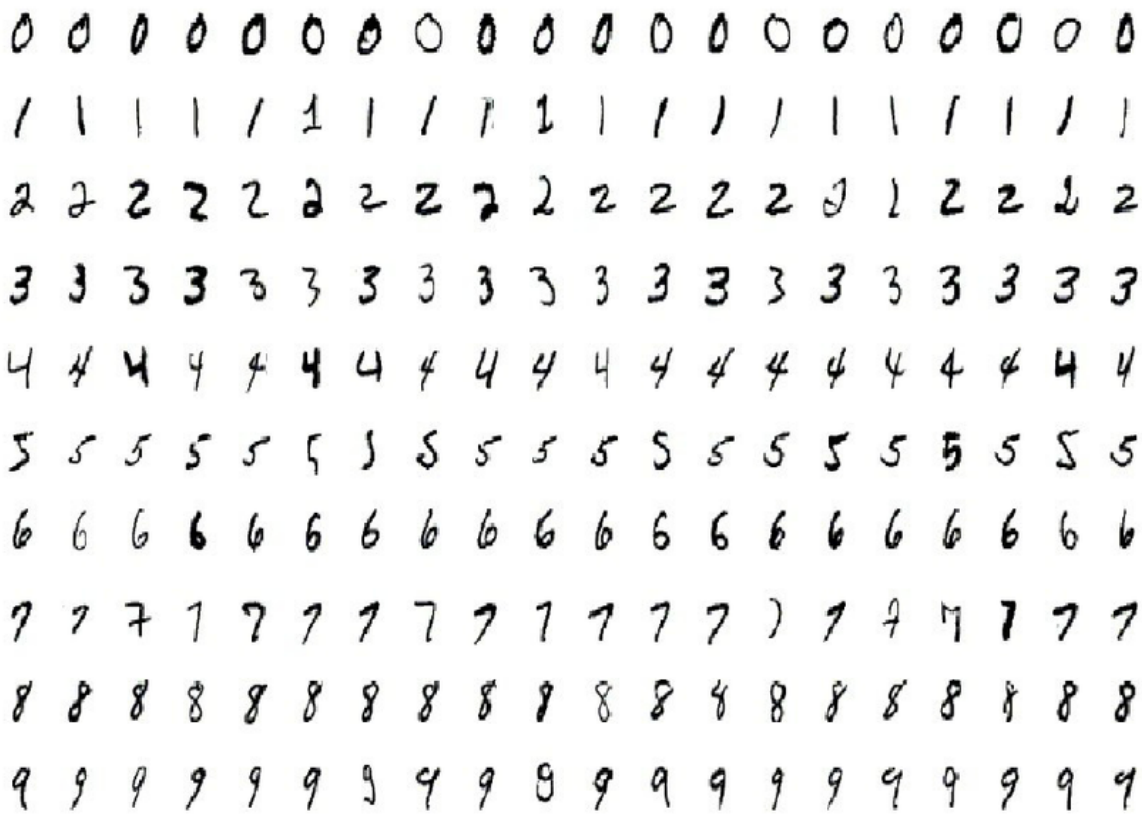


Fig. 3.3. Some examples of the images included in the MNIST dataset

Using the previously mentioned summary method on the training vectors for X on this particular dataset will not provide us with enough information to estimate the classification task complexity. As we can see from Table 3.2, we just know the range of values of every image (already studied in the definition of the dataset) and the mean, which only states that out training pixels tend to be white rather than black, the reason being that the characters are written over white paper. In this case, since all of our bits are grey values) , it is on our best interest to try to simply our task by dividing our training and test numbers into binary representations of their types. Achieving it will only mean to transform them into 1s and 0s, and easier representation to handle for our transformation block.

Variable name	Minimun	Median	Mean	Maximun
X training	0	0	33.32	255
X testing	0	0	33.32	255
Y training	0	4	4.454	9
Y testingg	0	4	4.443	9

Table 3.2. R SUMMARY METHOD ON MNIST.

If we instead observe the summary from the *Y* training data we can see that the dataset is very balanced, as the mean is close to half of the distance between the Minimum and the Maximum. This tells us that we will be able to probably achieve high degrees of accuracy on our classifier, as well as that the Entropy Triangle should also perform a satisfying job when analysing its informational flows.

Our goal when fitting MNIST through the autoencoder is to asses its capabilities when using bigger chunks of data and compressing it. Numerous reports and information available online give plethora of information about possible classification methods and their expected outcome in terms of error rates and accuracy.

3.1.3. Ionosphere

Include citation.

Ionosphere represents a set of data collected by a radar in Goose Bay, and later used for data science purposes by the Johns Hopkins University **citation**. The antenna had a phase array of 16 high-frequency antennas and used times pulses and pulse numbers for processing. The outputs can be labeled as either "good" or "bad", referring to the fact that a radar signal going through the ionosphere and thus showing no evidence of the existence of an ionosphere was labeled as "bad", and in any other case we would be labeling it as "good". It can be found on the R package "mlbench".

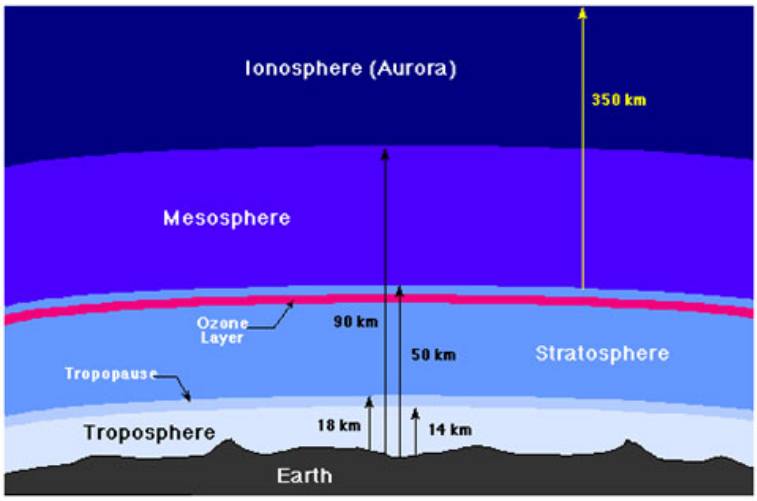


Fig. 3.4. The Ionosphere is one of the main layers composing Earth's atmosphere

There are 351 observations from 35 independent variables, with 33 of them being numerical values and 2 of them representing nominal values (one of them defining the class). However, one of the variables included in the dataset can be safely removed since it only represents a constant value (0). Having removed that one it should be remarked that this dataset is not balanced, since we don't posses the same number of "good" or "bad" class labels, and the difference is big enough to potentially affect its transformation and classification. We can expect its performance to be worst than the other two datasets mentioned previously. In the Ionosphere case, we will classify our data using binary decision, as there are only two states available for our labels.

Researchers have found very high accuracies when using non-linear perceptrons on this dataset [19], by using the same quantities of "good" and "bad" observations, which would not fit our needs. Although it can be easily inferred that we are going to get some degenerated results when compared with other papers, it seems like an appropriate choice to check the performance of the Entropy Triangle.

3.2. The classifiers

3.2.1. K-Nearest Neighbours

K-Nearest Neighbours (KNN) is a non-parametric supervised classification method used for estimating the density function [20]. As an supervised algorithm, it needs labelled

data to learn the appropriate function that produces new data belonging to the different regions of the existing classes when introducing new unlabelled data. Depending on the desired predictors we will want to obtain either a regression or a discrete output in our classifier.

The KNN algorithm hinges on the assumption that similar data must be close in a metric space representation. This hypothesis means that, in practice, we must take into account that proximity will be primordial to the outcome of our classification process when we are using the KNN classifier. This particular way of solving a data analysis problem is specially interesting when the data we are trying to analyse is very close to the same labelled data, since KNN relies in a "majority voting" scheme to predict the class for a sample using K nearest neighbour samples—hence the name of the technique—, where K is a fixed parameter of each run of the algorithm.

Although the majority voting classification method has some positive advantages **citation**, it also comes with some drawbacks too: in those cases where the class distributions are skewed, we can end up with a class which solely dominates the predictions. There are many ways of overcoming this issue, such as assigning proportional weights or to build clusters with similar points, but the reality is that KNN has a defined scalability that depends on the noise of your data and the characteristic of your dataset **citation**.

In order to try to improve results and polish the overall performance of the algorithm there are some tricks to take into account and follow when designing it:

- Trying to fit data through multiple instances of your implemented algorithm with new values for K in each case.
- Avoid using even K's values when doing binary classification, as we want to avoid tied votes.
- Perform an exploratory analysis of your data and design the boundaries and expected performance of your algorithm. Sometimes, realizing that a problem will be very lengthy and tedious to solve using a pre-determined method will save you a lot of time .

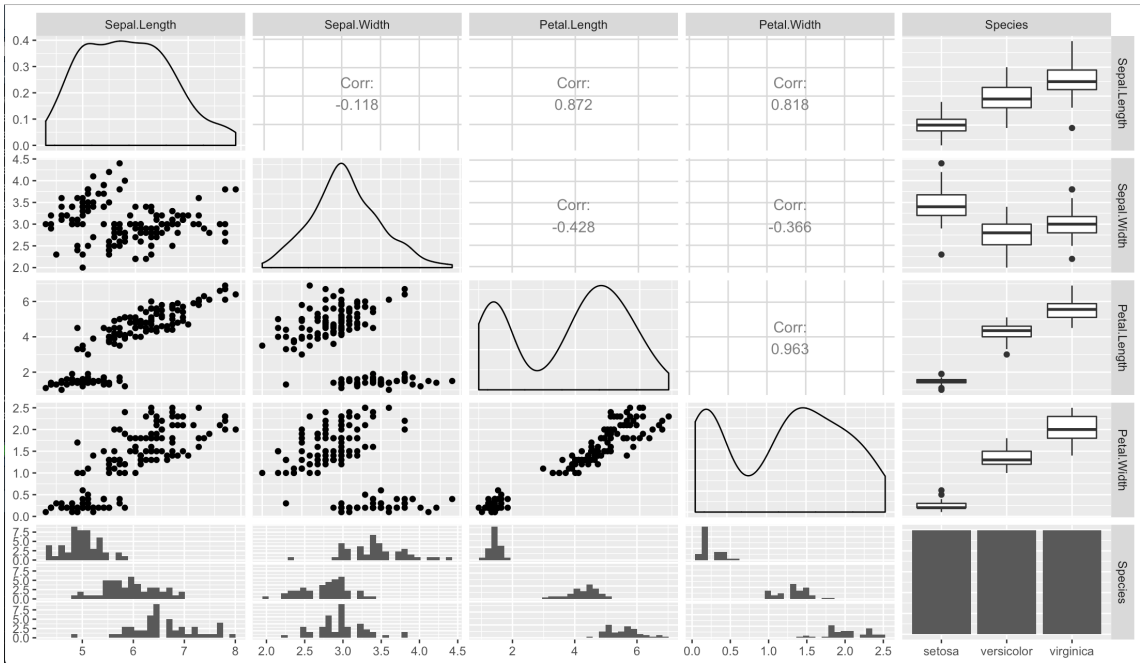


Fig. 3.5. The GGally R package also provides useful tools for this types of problems. Here, the correlation between Petal.length and Petal.Width is proven to be very high, as the plots from Figure 3.2 showed

3.2.2. Multilayer Perceptron

The Multilayer Perceptron (MLP) has a similar architecture to that of DNNs. Both of them have input, output and hidden layers as well as having nodes and using back-propagation for training. They even need their layers to be activated, as well as support the most common functions used in the Autoencoders. It can even be considered it as a slightly less computationally powerful DNNs.

It would seem as the main interest into MLPs could be to use them for smaller Autoencoders, but actually they possess a very important feature: they can distinguish and classify data that is not linearly separable.This property can help us qualify and compare different types of classification tasks [21]. As it is interesting to test multiple algorithms to make the most out of our ET, we can may compare them to the rest of the classifiers presented in this Section. It can be especially useful when we compare it with a KNN classifier, and we will implement both of them in our project to perform the same classification tasks over the same datasets. We would expect it to generally perform better on more complex datasets than KNN, which suffers when data is more difficult to differenti-

ate.

3.3. Principal Component Analysis

3.3.1. Description

Principal Component Analysis (PCA) is a statistical procedure based on orthogonal transformations to convert observations of possibly uncorrelated variables into sets of linearly uncorrelated variables called *principal components*. The transformation is performed in such a way that the first component has the largest possible variance, with each succeeding component having the highest variance taking into account that it has to be orthogonal to the preceding components. We will finally end up with a set of vectors which are uncorrelated. Due to the nature of the transformation, PCA can be affected by any type of scaling of the original dataset [20]. We also know that this method is valid to asses the capabilities of the ET due to the fact that it's use has already been introduced into experiments [15].

PCA is used in the data analysis field for Exploratory (EDA) or Confirmatory Data Analysis (CDA)[2] [22]. It can be used to visualize the genetic distance between populations of data. Results from the PCA are usually discussed taking into account their components. Other characteristics include:

- PCA is a simple eigenvector-based multivariate analysis. This is specially important to our task since this operation can help us discern the real structure of the data. It can also reduce the dimensionality of the data to view its most informative components and features.

- It resembles factor analysis **citation**. Both of them are used to describe the variability of the data and reduce its dimensionality, but use different techniques to reach that goal.

- It also resembles canonical correlation analysis (CCA) **citation**. While CCA tries to describe the cross-variance between two datasets, PCA defines the variance of a single dataset by using an orthogonal coordinate system.

[2] Modernly called "Predictive Data Analysis".

3.3.2. PCA and Information Theory

Our goal when using PCA in this project is to compare the results of the Autoencoder with another widely used tool in the data science field. At the same time, we want the theoretical underpinnings to be tacit on both methods used so that the outcome can be correctly compared when using the Entropy Triangle.

PCA fulfils our requirements by trying to minimize informational losses. If we assume our model vectors to be defined by the following equation :

$$\vec{x} = \vec{s} + \vec{n} \tag{3.1}$$

where $\vec{x}$ represents the vector being the sum of the desired information-bearing signal $\vec{s}$ and a noise (multivariate) signal $\vec{n}$. If we use this equation, it can be assumed that our vector can be dimensionally reduced.

For this model to hold true, the noise $\vec{n}$ has to be multivariate Gaussian with a covariance matrix proportional to the identity matrix. This fact allows us to maximize the mutual information between the dimensionally reduced output $\vec{y}$ and our $\vec{s}$ signal, only if we consider that the same assumptions made for $\vec{n}$ also apply for $\vec{s}$.

On the other common scenarios, on which $\vec{s}$ is not Gaussian, at least we will have an upper-bound for our representation such as,

$$I(\overline{X}; \overline{S}) - I(\overline{Y}; \overline{s}) \tag{3.2}$$

Recall that mutual information is defined on random vectors, in general, e.g. for $\vec{x}$ being a realisation of $\overline{X}$.

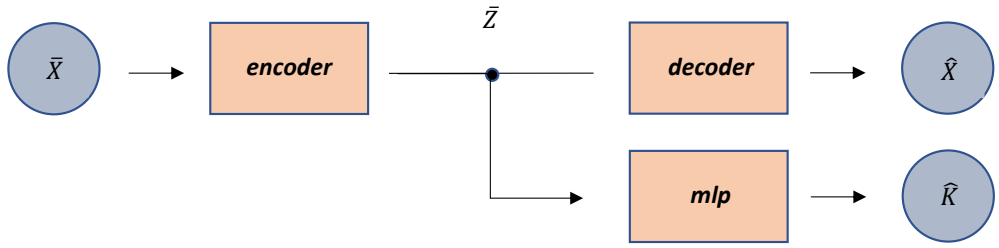
If our noise is dependent, the PCA looses its informational properties and thus we cannot use the previous representation. On the other hand, if our noise is more akin to a Gaussian than our information-bearing signal $\overline{S}$, our PCA representation will be improved.

3.4. Experiment setup

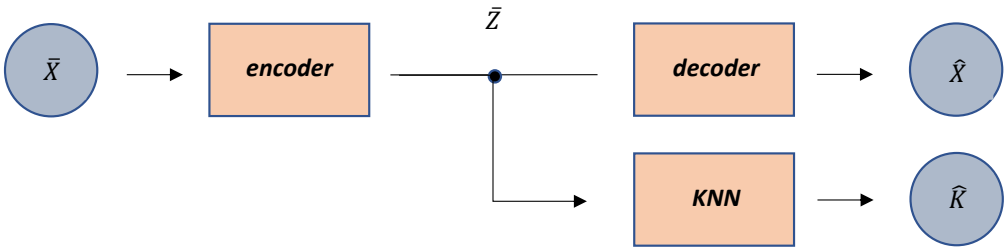
In this section we will be presenting the results regarding the different experiments performed over the datasets using the methods previously introduced. All the experiments

will have different sets of requirements and levels of complexity, what will help us discern the capabilities of the tools involved. To be able to do so, we have to establish the same structured tests on every dataset[3].

We retake here the training scheme introduced in Chapter 1 and presented in Figure 1.1.b where the *first* half of the Autoencoder, the actual encoder, is used as a mechanism to obtain an efficient representation $\hat{Z}$ of the input data $\hat{X}$. We will use this representation as the input for the classifier. In the following we use both a KNN and a MLP classifiers, as depicted in Figure 3.6.



(a) Autoencoder and MLP.

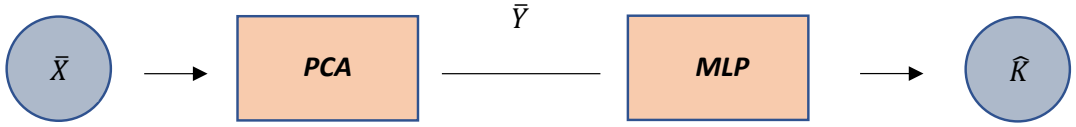


(b) Autoencoder and KNN.

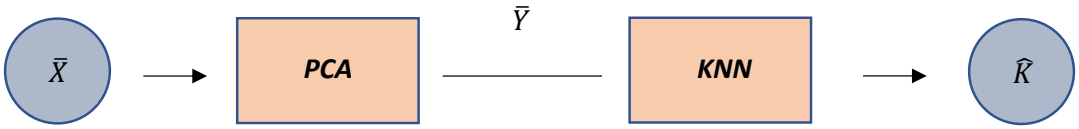
Fig. 3.6. Structure of the Autoencoder and it's classifiers

We want to compare the effectiveness of the Autoencoder and PCA (see Section 3.3) to obtain a "good" representation for the input. We can see in Figure 3.7 that we use the output from the PCA as the input of the same classifiers as before with the aim to be able to compare their results.

[3]Due to the impossibility of using `caret::train` on MNIST (Every laptop I tried to use for this purpose would just get stuck and force the closing of RStudio), we will be not be using Knn to compare results on PCA and the Autoencoder.



(a) PCA and MLP.



(b) PCA and KNN.

Fig. 3.7. Structure of the PCA and its classifiers

To increase the reliability of the results we will use k-Fold validation [20]. This is a technique based on dividing a dataset into k disjoint subsets of samples—or "folds". We are interested in measuring how each of the architectures performs in each fold of our datasets when training or "fitting" the classifiers involved in the other $k-1$ folds. By doing so we obtain—on less training data, that is the $k-1$ folds—an estimate of the performance of the system involved in the retained fold; by repeating this over all of the folds we can also estimate an average and the dispersion of these measures. Importantly, we can use this technique *on the whole data* at the cost of a slightly increase of the data-dependence of the results. Using $k = 5$ means that we use 5 times 80% of the data for training and 20% of the data for testing.

Our hypothesis is that each fold experiment will provide us with similar data and results in our plots when we use balanced datasets. However, if we have unbalanced datasets we expect to see more inconsistency through them.

The Iris dataset experiment will be thoroughly described and explained as we will be using it as an example of to perform the methods and theory explained on the previous chapters of the report. The remaining experiments will avoid unnecessary explications regarding concepts re-used from the Iris experiment.

At the end of each section, the overall performance of each dataset will be tested with the Entropy Triangle, which will be used to compare the entropy between the data of each Fold and the output of used classifier.

3.5. Iris Dataset

3.5.1. Data Preparation

In order to train the data through the Autoencoder, we will firstly have to transform it into an easier shape which will reduce the difficulty and length of our task. Normally, Deep Neural Networks will do a good job as long as the data has a reasonable scalability, so applying a normalization to our data is not mandatory. However, it is easier for us to use a simple pre-processing in our data, which given that the observations from the variables have different ranges of values will help us reduce the training time of our Autoencoder.

Although as we can see from Figure 3.1 and Table 3.1, some of its features are closer to what it could be referred as a Normal distribution, specially Sepal Length and Sepal Width. Other variables, such as Petal Length, are a little bit off and far away from that description. To be able to fix those disparities, we will apply a Box-Cox transformation, which consist basically of applying the following equation on your data:

$$y(\lambda) = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda}, if\ \lambda \neq 0 \\ \log y, if\ \lambda = 0 \end{cases} \tag{3.3}$$

Where λ is an exponent with a value inside the range $[-5, 5]$. The Box-Cox method will automatically assign the value of lambda that fits your dataset the most. This value is selected simply by looking for the optimum value by fitting the equation 3.3 .

Once we apply this function, we can start to see the results on our newly created data by checking its pairs function again:

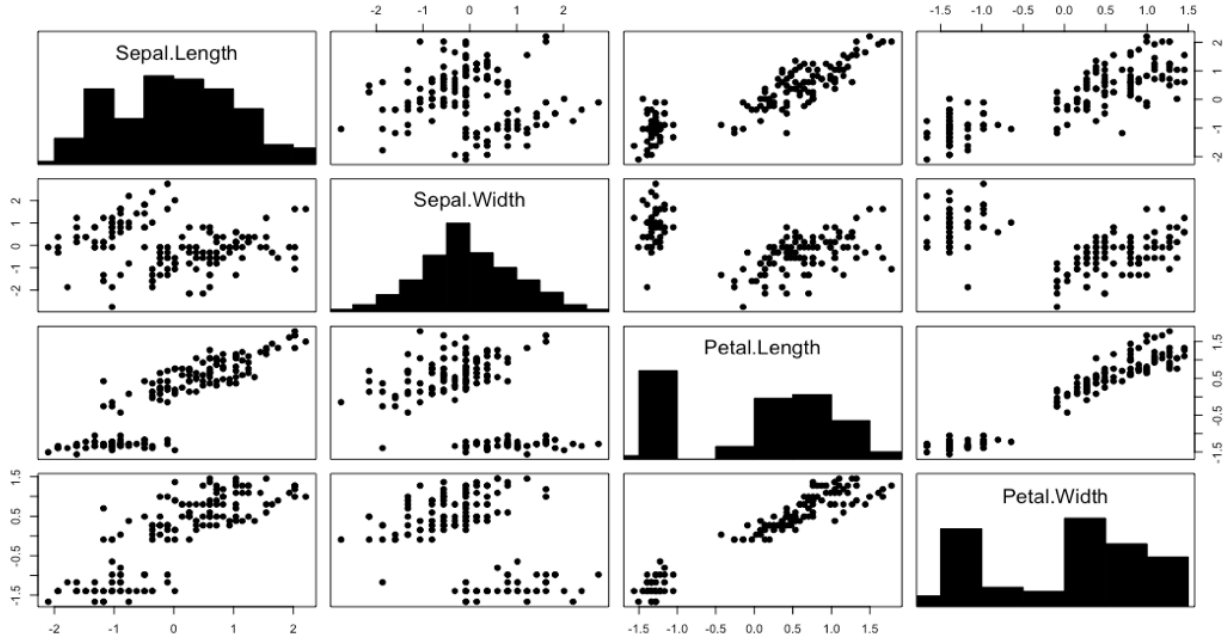


Fig. 3.8. Using the pairs function on Iris Boxcox

Which has an histogram slightly different to the one presented on Figure 3.8, but we can really see the new features of our transformation if we use use the summary function and compare its results with respect to the ones obtained before on Table 3.1

Variable name	**Sepal Length**	**Sepal Width**	**Petal Length**	**Petal Width**
Minimun	-2.10	-2.75	-1.56	-1.66
Median	0.02	-0.08	0.33	0.27
Mean	0.00	0.00	0.00	0.00
Maximun	2.20	2.75	1.78	1.45

Table 3.3. R SUMMARY METHOD ON IRIS BOX COX.

By looking at the Table on top, it can be seen that as expected the Mean of our distribution switched from it's previous value to 0, essentially telling us that the transformation has been successful. It is also important to point out that although the values of the vector that are in our dataset are different, we still hold the same proportions and plots of the observations, which means that the general information that each of our variables contained

its still there. We just shaped it differently so that it is easier for us to work and perform operations with them.

3.5.2. The Autoencoder

For the Iris dataset, we will use a simple Autoencoder, with just a few layers, that should be enough to complete the training required. As our given dataset doesn't contain a lot of observations, this approach to the Autoencoder aims to show an example of use. That is why, if we take into account that we only have four variables to compress, we can see that the maximum compression we can achieve is a 25 % of the original size. But, in order to be sure we have a working architecture, I will be using just a 75 % compression in the middle layer. On the other hand, our model will have a data expansion of about a 400 %, so that we are able to do a real compression in each of the subsequent layers.

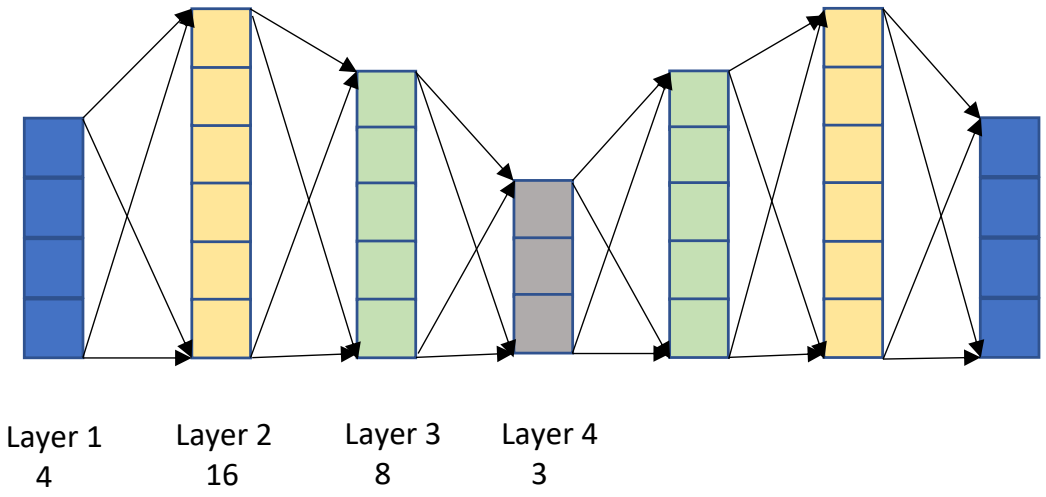


Fig. 3.9. Architecture of the Autoencoder in Iris

On Figure 3.9, each of the layers has the size assigned below it. But before choosing the compilation method we are going to use for the Autoencoder, we have to choose the activation functions for each layer. In this case, we decided that the Relu function would be used **for all the layers of the Autoencoder except the last one**, where we will use a Sigmoid function. We decided to use this setup because the Relu function does a good job at selecting the informational qualities of each component while at the same time keeping

the training time of the architecture very low due to the simplicity of its definition, while the Sigmoid function is more complex and thus seems to be suited for the last layers and to finalize the transferring of information.

The last thing to take into account is the compilation of the whole structure created with the previously mentioned Autoencoder. Here, we will have to choose between the wide range of available options on the keras package in R, to try to maximize the performance and optimization of our model. To do so, we have to first rule out all the loss functions that don't apply to our task. For example, although used in a lot of DNNs, the categorical functions don't fit our criteria since we are not performing a classification task here. Moreover, our range of data is not between 0 and 1, which would make this even a worst choice. For us, since we also are trying to predict the output data using some training data, we can assume we have a **regression problem** in our hands.

Having established that, we have a narrower set of possible choices to make. Between all of the left available ones, we decided to use the **Mean Squared Error function**. The MSE is calculated as the average of the squared differences between the predicted and the original values. This means that the bigger the difference, more punitive this metric is for our model. This works great for the intent of our experiments, since we want our model to be able to get very accurate predictions from the compressed version of our data.

Finally, we also need to decide on our optimizer for the compilation. Here we decided to narrow our options to chose only between the Adam and the Adadelta. Finally, it was seen that the Adam optimizer was achieving the same level of losses as the Adadelta, while needing less epochs. Due to the amount of passes that we have to do through the Autoencoder, it is useful for us to get reliable data in the shortest time possible, so it was finally implemented using Adam.This decision would allow us to help mitigate the problem of time, which can be quite demanding when dealing with multiple transformations and neural networks.

3.5.3. The Classifiers

Knn on PCA and Autoencoder

Once we have obtained the output of our Autoencoder, we have a resulting data frame with the same number of observations but a reduced number of variables, which in the case of the Iris Autoencoder will be three, making our resulting matrix to have a shape of 120×3. It also must be notest that the process is the same for both the output of the Autoencoder and the PCA.

The matrices resulting from this compression process may not have all of their columns or rows with a value different from 0. This does not pose a threat to our training, since none of the methods chosen for our training will drop our results. In the case of the Knn we will get some Warnings, but as we will see later it won't affect the result of the process.

```
k-Nearest Neighbors

120 samples
 3 predictor
 3 classes: 'setosa', 'versicolor', 'virginica'

No pre-processing
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 120, 120, 120, 120, 120, 120, ...
Resampling results across tuning parameters:

 k   Accuracy   Kappa
 5  0.8650582  0.7956975
 7  0.8561286  0.7828639
 9  0.8620067  0.7915562
      ........          .......
33  0.7444733  0.6203610

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
k-Nearest Neighbors
```

Fig. 3.10. Knn example on Iris

The function that we decided to use for our Knn training was provided in the train method in the caret package. It only requires you to specify the training data, labeled

as x, the labels to be trained with, named as y, and the method, which is a simple Knn classifier. Once provided, the function will start to train your model.

Once the model has finished training, we will use the model generated with our values to predict the labels of our model and compare it to the labels used to train it on the first place. Once we have calculated that, we will use the confusion Matrix function to asses the accuracy of our predictions when compared to our original labels. A Confusion Matrix is a table with the same number of columns and rows as your class labels, being the only difference that the diagonal represents the predictions from your model and the class labels that matched, while anything outside of it accounts for the number of missed predictions with respect to the class. In this case, this method will provide us with a 3×3 matrix which we then can feed to the jentropies method.

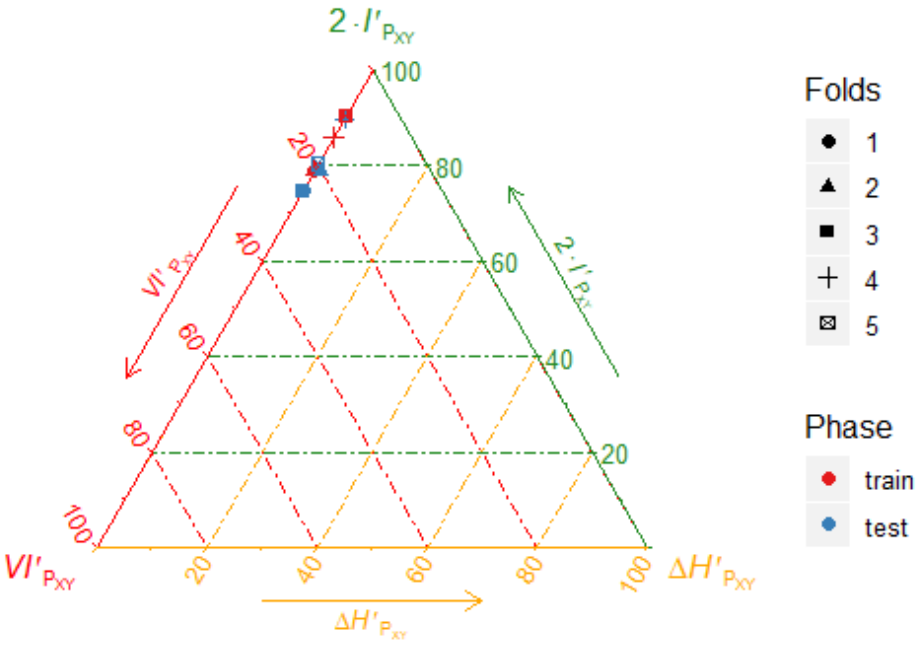


Fig. 3.11. Entropy Triangle using Autoencoder + Knn in Iris

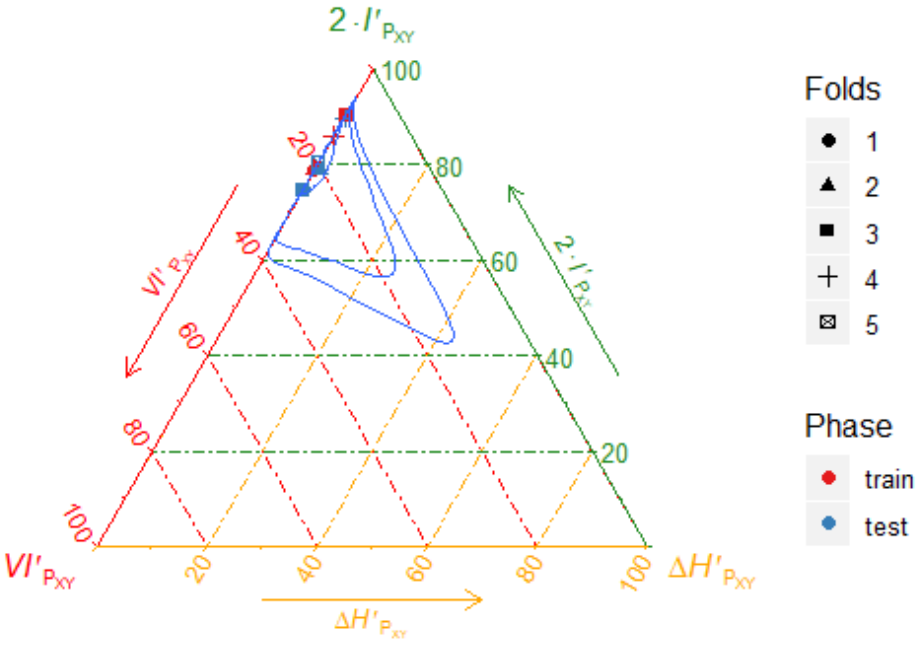


Fig. 3.12. Entropy Triangle confidence interval using Autoencoder + Knn in Iris

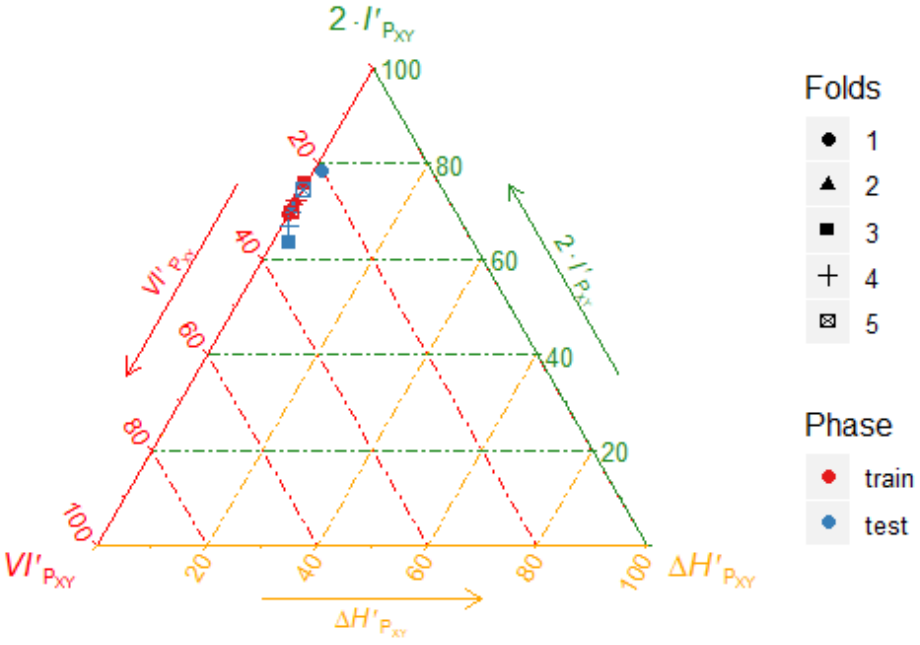


Fig. 3.13. Entropy Triangle using PCA + Knn in Iris

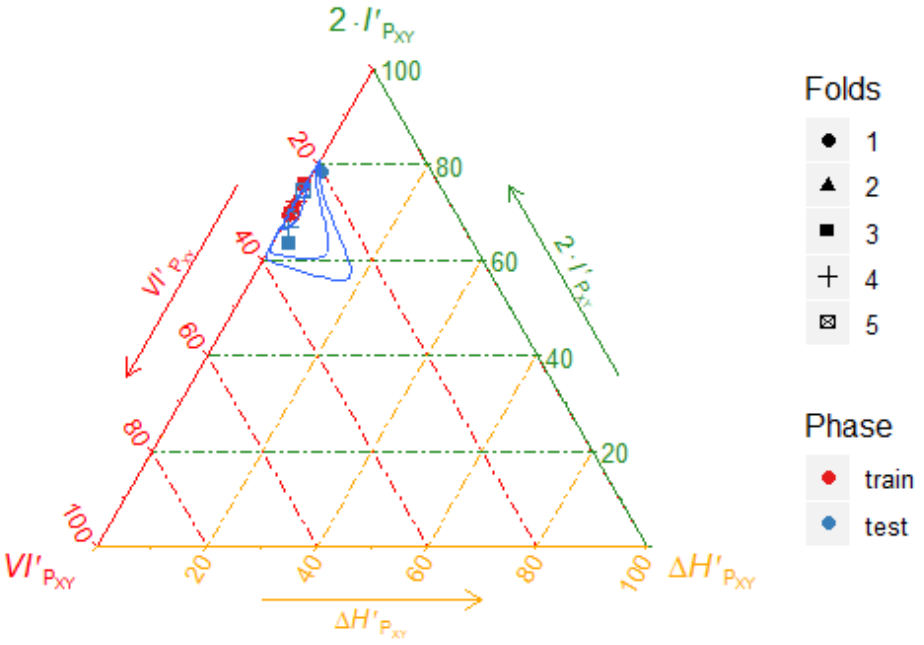


Fig. 3.14. Entropy Triangle confidence interval using PCA + Knn in Iris

Given that we are using a K-Folds validation with a value of $K = 5$, we are plotting 5 coordinates for our test and train variables, just to see how well our model performed.

From Figure 3.11 we can see that the values are spatially close inside of the Entropy Triangle. This leads us to believe that the training has been accurate. We usually tend to expect the training and the testing folds to be close if the training process has a balanced set with distinguishable traits differentiating each class from the rest. From the coordinates plotted in the Triangle we can also see that our transformation and classification process has been successful at transmitting the information. By following the guidelines from 2 we have a balanced classifier predicting accurately the labels of our classes.

From Figure 3.13 we can asses that the conclusions drawn from the Knn of the Autoencoder can also be applied for the PCA. Both figures show a similar behavior, although the Autoencoder has slightly better results as its folds representation is situated closer to the peak of the ET.

In addition to plotting the entropic characteristics of the variables, it could be also interesting to take a look at Figure 3.12 and Figure 3.14, which represent the Confidence Intervals of the represented points in the Entropy Triangle. By observing it, we can see

that the confidence interval for the Knn in the Autoencoder is spatially greater than the PCA one, leading us to believe that the interval of possible values for the Autoencoder is greater and thus subjected to greater changes in the prediction, probably due to the training uncertainty of the Autoencoder and its effect on the value of $\overline{Z}$.

MLP on PCA and Autoencoder

Before fitting the data through a MLP, we have to specify the structure that we are implementing on our Neural Network. Seeing that the PCA and the Autoencoder both predict different lengths for our target training matrices, we have to take into account that the input shape will differ.

Once we know that, we can start looking at our structure. Since we are dealing again with a small number of variables, a simple approach will be to divide our MLP into 4 layers. Firstly, we place an input layer with the size of our data, either a 3×120 in the Autoencoder or a 4×120 in the PCA. Secondly, we can place an expansion layer of size 5, followed by the first compression layer of 4, which is connected to the last one which will be the output of our architecture.

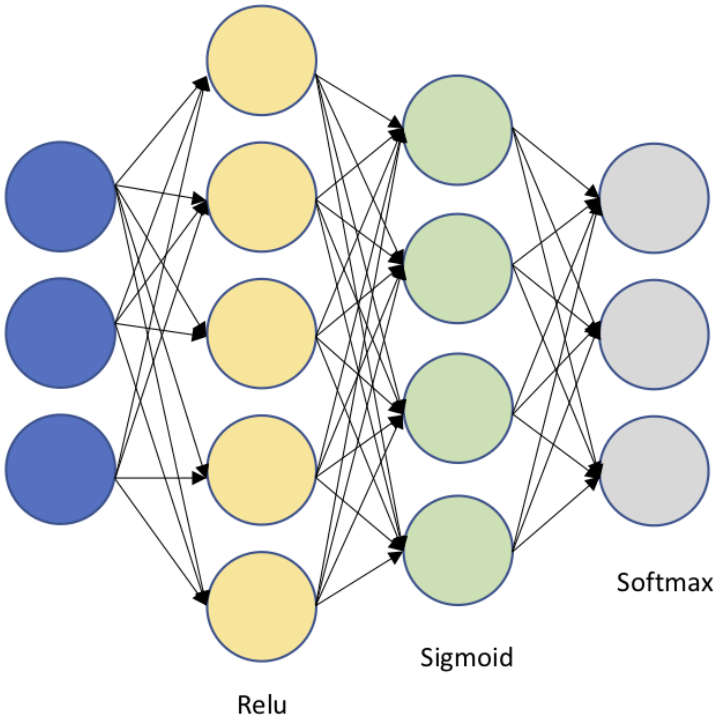


Fig. 3.15. MLP architecture in Iris for the Autoencoder

After figuring out the structure, we now have to decide which activation function would fit the best for each layer. As in the Autoencoder, the first layer will be a Relu, the second one acquires the role of the Sigmoid and finally, as the MLP is a classifier, we have to place a softmax activation function at the end. We want the output of our Autoencoder to be a decision and the softmax is the chosen solution, as we will get our output variables to have an added value equal to 1, being the greater one the label predicted.

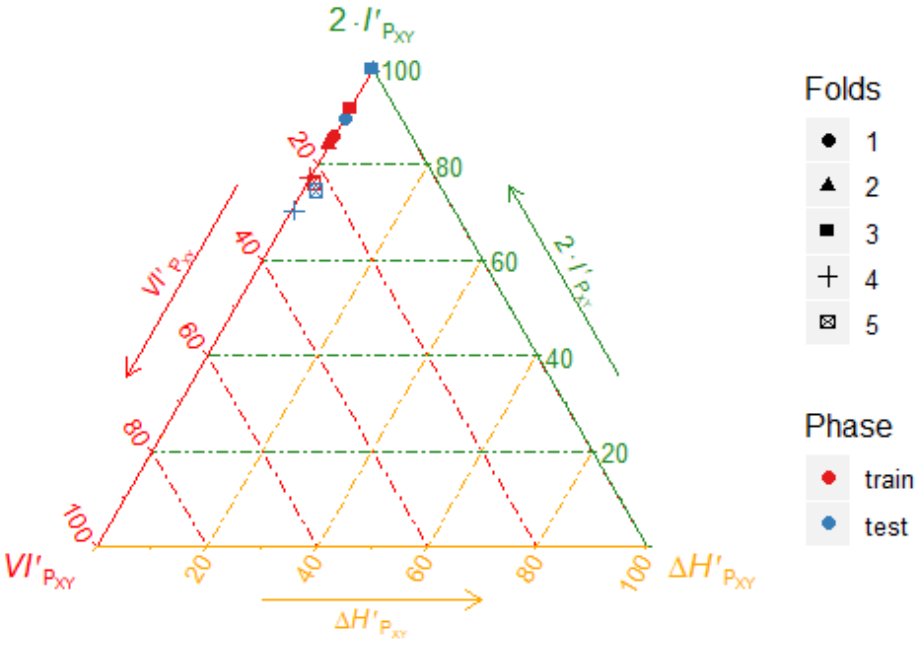


Fig. 3.16. Entropy Triangle using Autoencoder + MLP in Iris

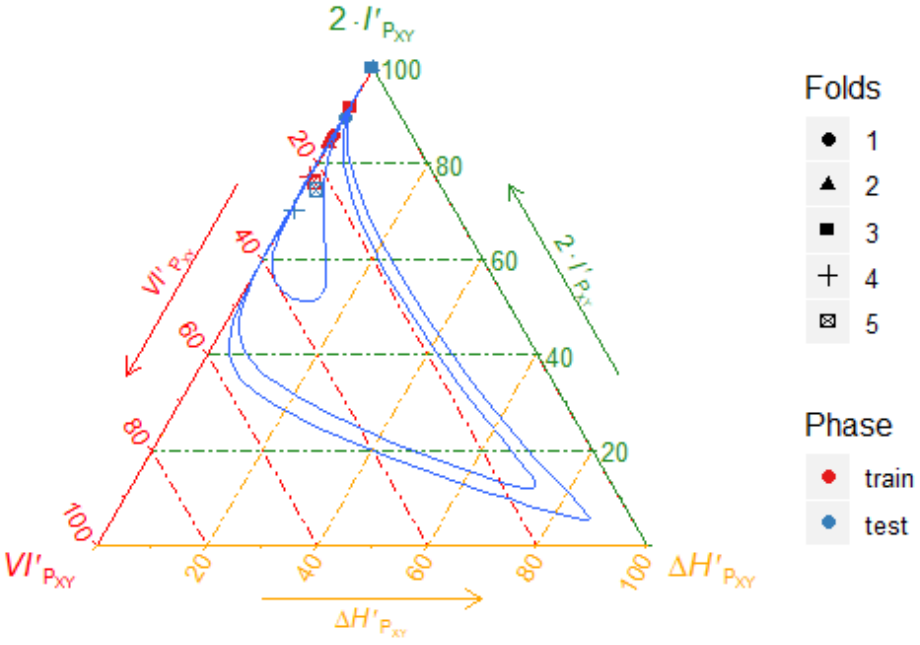


Fig. 3.17. Entropy Triangle confidence interval using Autoencoder + MLP in Iris

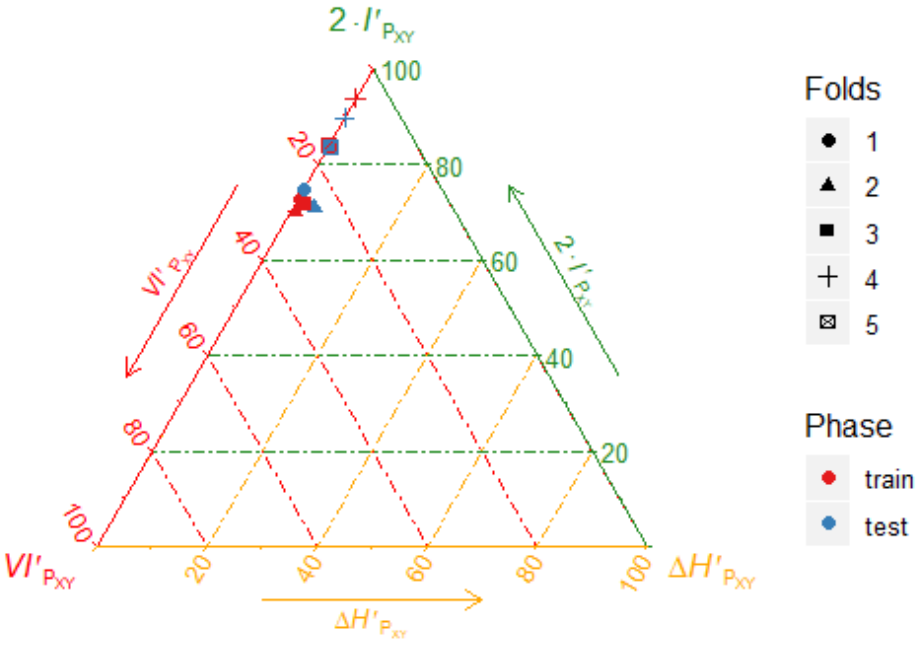


Fig. 3.18. Entropy Triangle using PCA + MLP in Iris

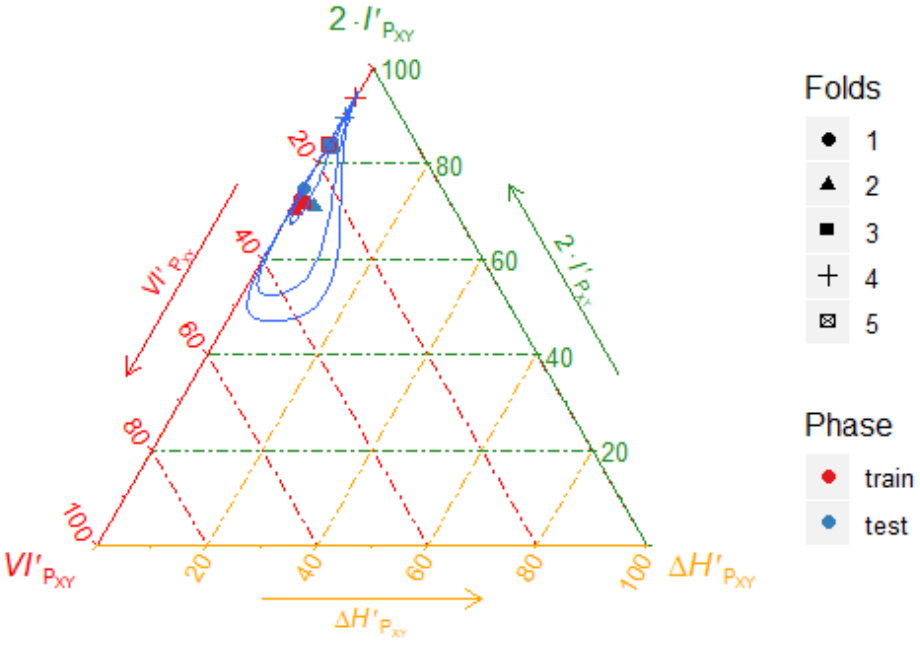


Fig. 3.19. Entropy Triangle confidence interval using PCA + MLP in Iris

From Figure 3.16 we can see that we have obtained more spatially separated points in our ET with respect to Figure 3.11 from the previous section. Nevertheless, it seems that the results from the MLP classifier overall show a better performance at classifying

our observations than the Knn for the same $\overline{Z}$. We even see that the third testing fold was classified perfectly.

We observe the same spatial separation and behavior on the PCA transformation by comparing again Figure 3.18 with 3.13. Again we are obtaining better results than those from the Knn.

On the other hand, Figure 3.19 and Figure 3.17 have greater confidence intervals than those presented in the previous section. The uncertainty introduced in the predictive process affects this time both the PCA and the Autoencoder, although we see that the training of the Autoencoder added to the training of the MLP has made our confidence interval in Figure 3.17 increase greatly. Basically, we can see here that without proper training our folds could specialize in predicting certain class rather than being balanced.

3.5.4. Results discussion

After analyzing the Knn and MLP by on its own, we now are going to plot the summation from all of the testing folds in order to get an overall idea about how well each one of our transformations + classifications performed.

We are using the testing fold only in order to see how well our transformations performed in our data we have to compare the accuracy of its predictions over the whole dataset. We don't need to asses the training as we have already seen on the previous sections that our training folds results in the ET have been consistent with its testing observations.

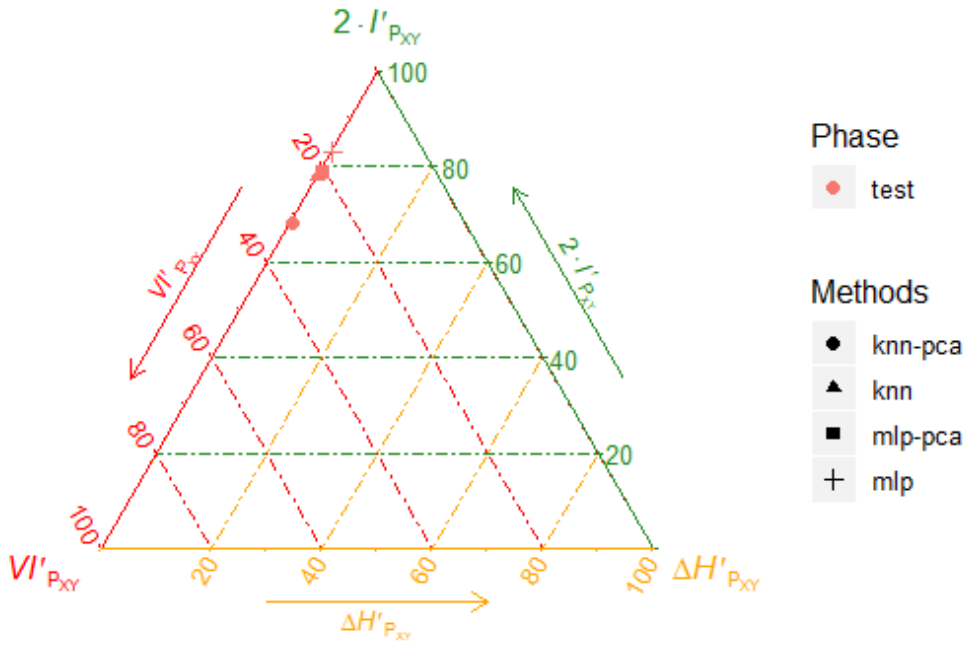


Fig. 3.20. Entropy Triangle in Iris with the total testing result

By looking at Figure 3.20 the MLP classifier on the Autoencoder seems to have outperformed the rest of the testing structures. The worst one is the Knn in PCA, which is placed lower in the ET. If we consider the reliability of all the classifiers included, we can see that overall all of them do a good job at classifying the Iris datasset. However, the accuracy of the MLP with the Autoencoder outperforms the rest of them.

3.6. Ionosphere Dataset

3.6.1. Data Preparation

The only pre-proccessing done for Ionosphere was explained in Subsection 3.1.3 referred to the datasets. No Box-Cox or other method was found to improve the performance on this particular case. We will only get rid of the column with no variance, which as mentioned before was a column with a constant value of 0 all throughout the set. As explained before, we expect this dataset to show more unbalancing in its results.

3.6.2. The Autoencoder

Ionosphere has more variables that Iris, but it is still a small Autoencoder compared to the structure that we are using for MNIST. Our architecture transforms the data provided by the package into 8 observations. We again are using an Autoencoder with the same amount layers structured as:

Number of Layers	Size	Activation Function
First	33	None
Second	50	Relu
Third	20	Relu
Fourth or Middle	8	Relu

Table 3.4. IONOSPHERE AUTOENCODER LAYERS OF THE ENCODER.

Number of Layers	Size	Activation Function
Middle or First	8	Relu
Second	20	Relu
Third	50	Relu
Fourth or Output	33	Sigmoid

Table 3.5. IONOSPHERE AUTOENCODER LAYERS OF THE DECODER.

As seen of both Table 3.4 and Table 3.5, with the aim to compare the results provided by both datasets, we are going to use the same compilation options and activation functions as in Iris. We think it would be interesting to test the same compilation environment for two datasets with completely different characteristics.

It is expected for Ionosphere to not be difficult to train, since it is very unbalanced and classifiers tend to just choose the predominant class in the dataset as their predicted value

in most of the cases. Both the MLP and the KNN suffer from this unwanted tendency, but we want to reproduce that behavior on this experiment.

3.6.3. Knn on PCA and Autoencoder

We will follow the same process to present the results regarded from the Ionosphere dataset:

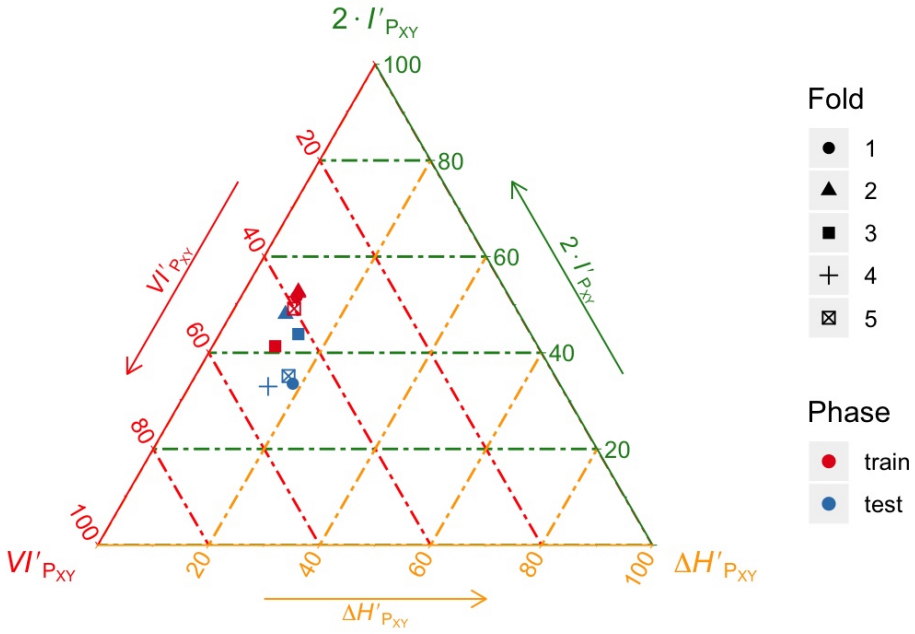


Fig. 3.21. Entropy Triangle using Autoencoder + Knn in Ionosphere

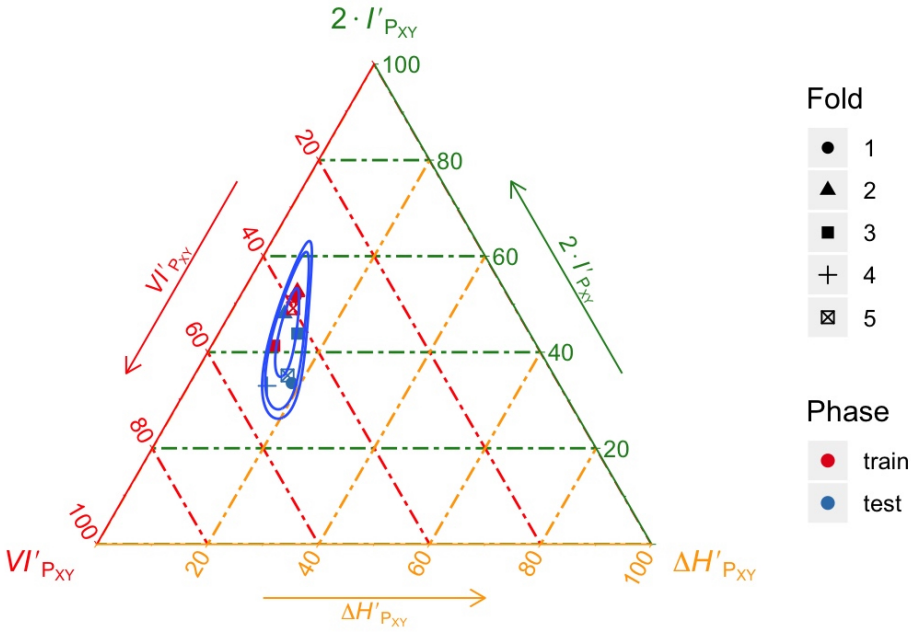


Fig. 3.22. Entropy Triangle confidence interval using Autoencoder + Knn in Ionosphere

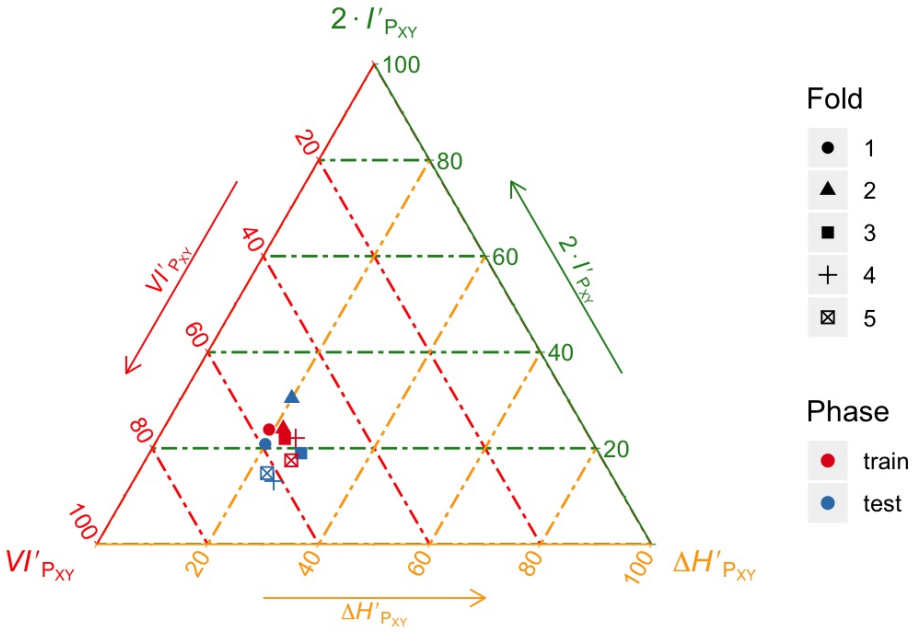


Fig. 3.23. Entropy Triangle using PCA + Knn in Ionosphere

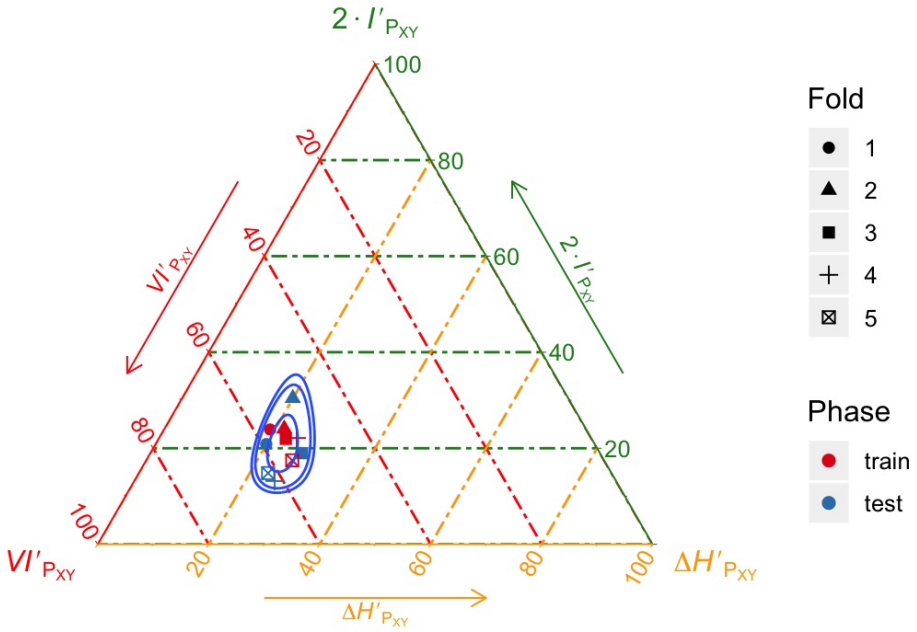


Fig. 3.24. Entropy Triangle confidence interval using PCA + Knn in Ionosphere

Just by taking a simple look over Figure 3.21 and Figure 3.23 it is noticeable that the unbalancing of the dataset has affected the outcome of our experiments. The coordinates of the entropies from our Knn Folds tells us that our predictions have been inaccurate and the transmission of information either by means of the Virtual Information or the Mutual Information have degraded a lot from the results gathered from the Iris dataset.

On the other hand, Figure 3.22 and Figure 3.24 point to us that sparsity of possible new test values is not very big, which basically is indicating that our classifications can't greatly vary their inaccurate predictions.

3.6.4. MLP on PCA and Autoencoder

To completely asses the Ionosphere dataset classification process we must now asses the performance of the MLP:

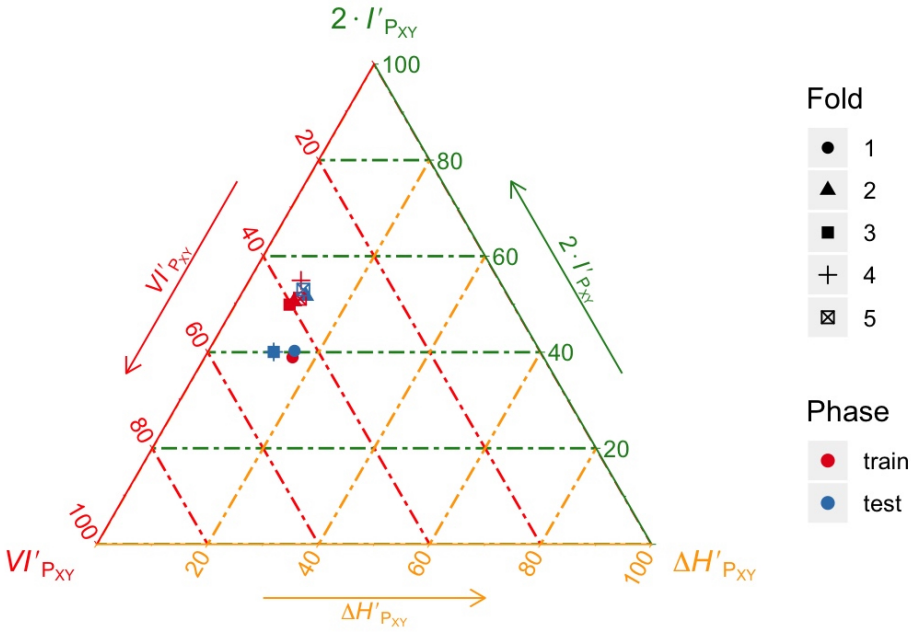


Fig. 3.25. Entropy Triangle using Autoencoder + MLP in Ionosphere

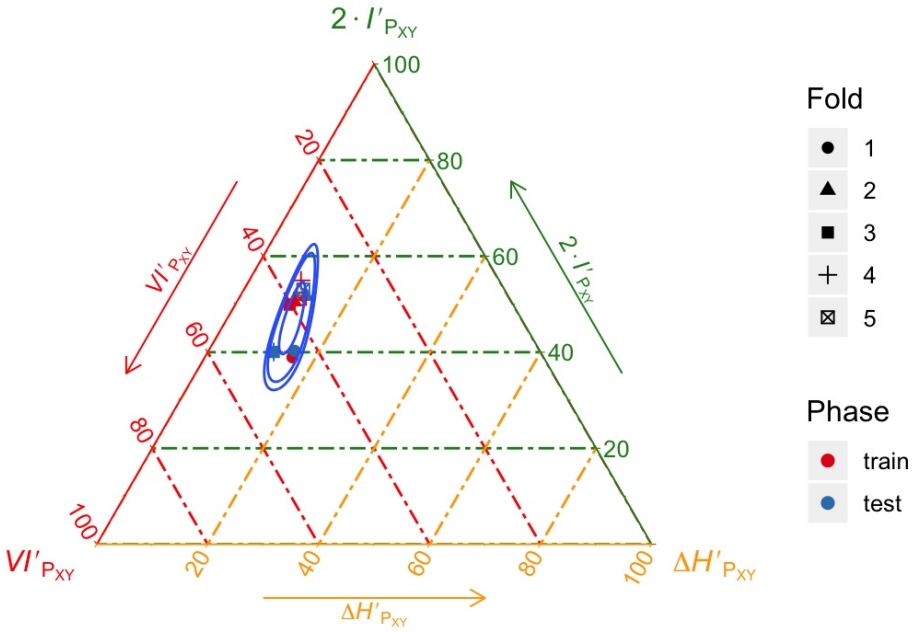


Fig. 3.26. Entropy Triangle confidence interval using Autoencoder + MLP in Ionosphere

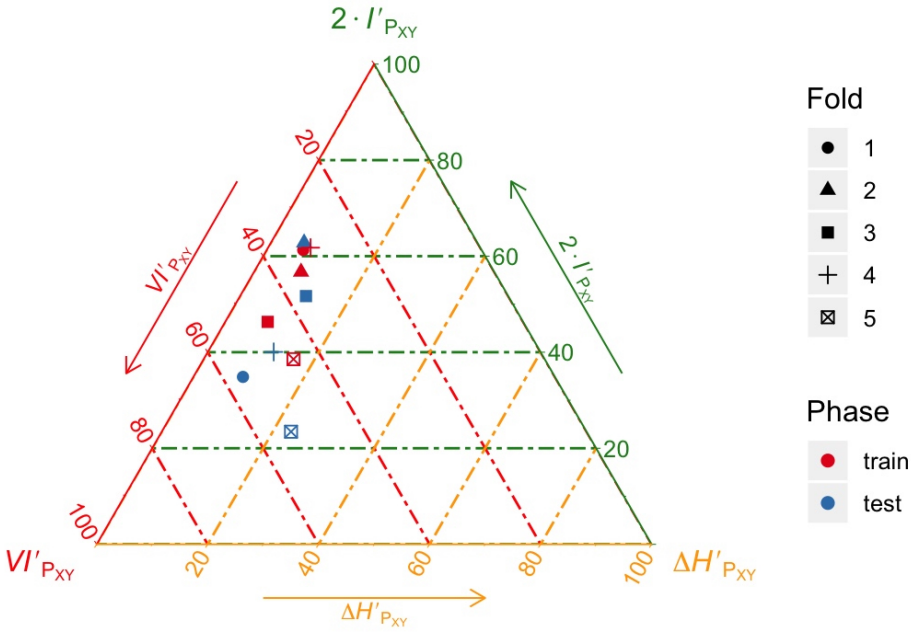


Fig. 3.27. Entropy Triangle using PCA + Mlp in Ionosphere

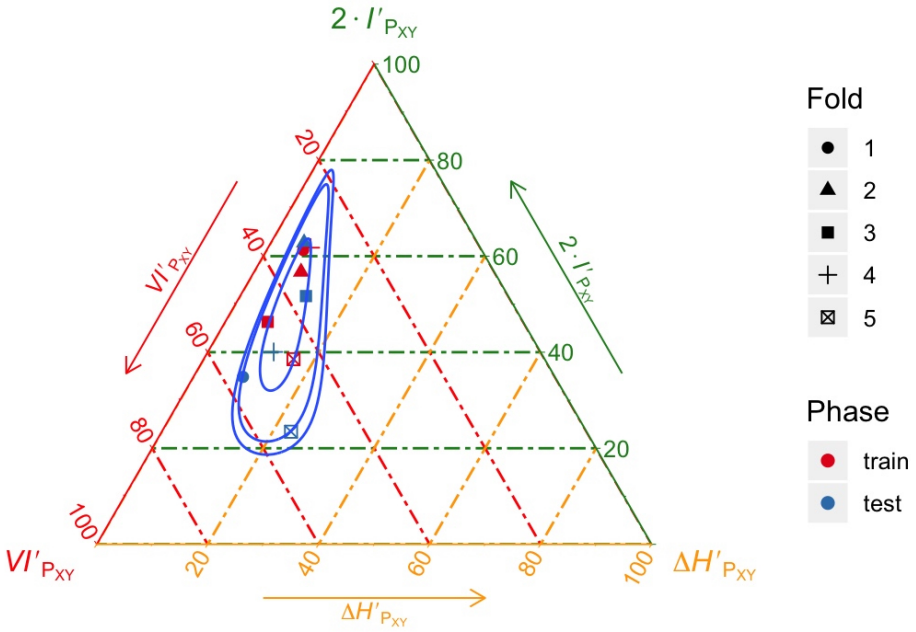


Fig. 3.28. Entropy Triangle confidence interval using PCA + Mlp in Ionosphere

As seen on the Iris dataset, the sparsity of our points is greater when using the MLP. This behavior is clearly recognizable in Figure 3.25 and Figure 3.27. We can also see that the degradation of results when compared to the Iris dataset is noticeable. Both the PCA

and the Autoencoder don't seem to outperform each other by a great margin. They do seem to improve the outcome of the predictions from the Knn, but the unbalancing of the dataset is affecting its performance.

3.6.5. Results discussion

If we want to further analyze the best classifier + transformation we must once again plot the total testing fold coordinates:

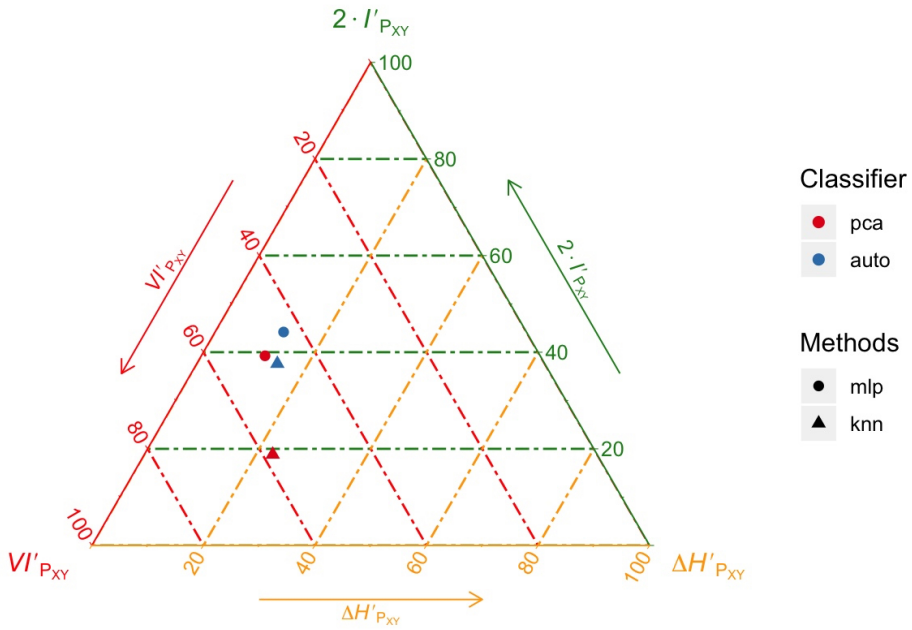


Fig. 3.29. Entropy Triangle in Ionosphere with the total test fold value

In Ionosphere, Figure 3.29 shows that the Autoencoder is more reliable to classify Ionosphere as both the MLP and the Knn seem to be close to each other in the ET, while the differences in the PCA are too great to consider that the Knn can be used to classify Ionosphere.

In the particular case of Ionosphere, it is interesting to take a look at the confusion matrices from the folds, as well as the accuracy of the training. We can also use some of the data acquired from the training of Iris to compare them.

For example, we can compare the third fold from the MLP in the Autoencoder from Iris and in Ionosphere we are taking the first fold from the MLP in the PCA:

(a) IRIS CONFUSION MATRIX

Accuracy = 93 %

10	0	0
0	10	0
0	9	10

(b) IONOSPHERE CONFUSION MATRIX

Accuracy = 94 %

18	3
7	42

Table 3.6. COMPARISON OF IRIS AND IONOSPHERE CONFUSION MATRICES

From analyzing Table **??**, we get more or less the same accuracies on both instances of the folds, but only in case of the Iris MLP we are getting results with that level of certainty. The Ionosphere dataset is so unbalanced and that its observations, regardless of the level of accuracy shown, are forcing errors on our predictor. This is an example of the **accuracy paradox** mentioned on **??**.

3.7. MNIST Dataset

3.7.1. Data Preparation

All the necessary processes needed to transform our data into a valuable representation of it for the purpose of the Bachelor Thesis have been explained on Section 3.1.2.

3.7.2. The Autoencoder

The Autoencoder used for the MNIST dataset has the most complex structure in terms of computational power out of all of the considered implementations for the sake of this thesis.

Number of Layers	Size	Activation Function
First	784	None
Second	1000	Relu
Third	500	Relu
Fourth	250	Relu
Fifth or Middle	64	Relu

Table 3.8. MNIST AUTOENCODER LAYERS OF THE ENCODER.

Number of Layers	Size	Activation Function
First or Middle	64	None
Second	250	Relu
Third	500	Relu
Fourth	1000	Relu
Fifth	784	Sigmoid

Table 3.9. MNIST LAYERS OF THE DECODER.

The Autocoder has a large amount of observations (Around 287.1 megabits only for the X training component), so we expect the training time to be longer than the previous datasets. We are also going to keep the same configuration for the activation functions of the architecture, as the other Autoencoders have proven successful to the transference of information.

3.7.3. MLP on PCA and Autoencoder

The structure implemented for the MLP in the Autoencoder and in the PCA will be completely different. As seen in both Table 3.8 and Table 3.9, the size of our variables is 64 in the $\hat{Z}$ from the Autoencoder, but the PCA creates a matrix with the same measures as the input $\hat{X}$, which makes us have to increase the size of our MLP greatly, thus making us add an extra layer to improve it's performance.

Number of Layers	Size	Activation Function
First	64	None
Second	128	Relu
Third	64	Sigmoid
Fourth	10	Softmax

Table 3.10. MNIST MLP LAYERS OF THE AUTOENCODER

Number of Layers	Size	Activation Function
First	1000	None
Second	500	Relu
Third	125	Relu
Fourth	64	Sigmoid
Fifth	10	Softmax

Table 3.11. MNIST MLP LAYERS OF THE PCA

As we can see on Table 3.11 the PCA MLP has the advantage of being composed by a greater amount of neurons, making it more computationally powerful and thus having an advantage towards the smaller Autoencoder MLP.

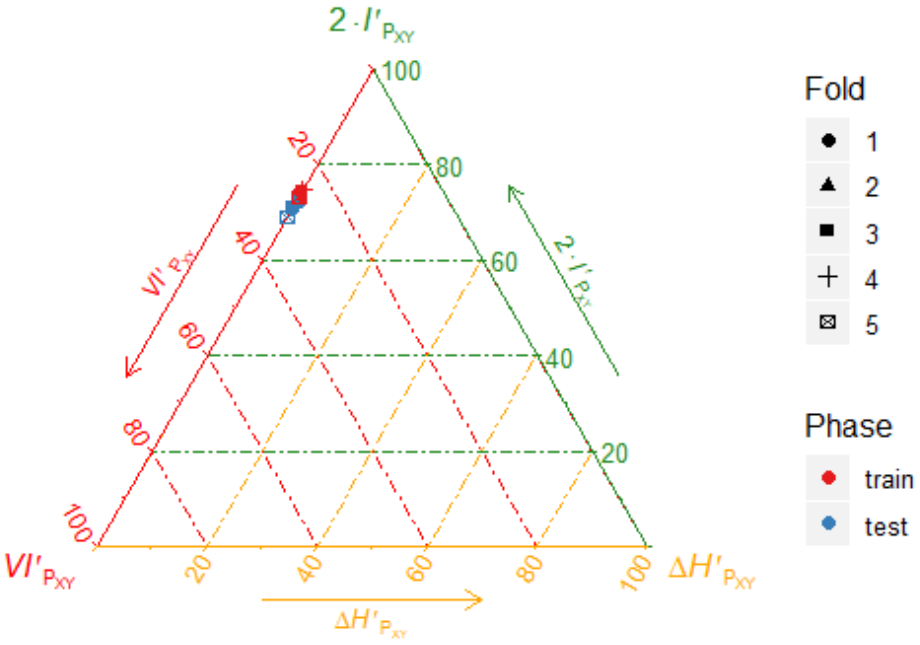


Fig. 3.30. Entropy Triangle using Autoencoder + Mlp in MNIST

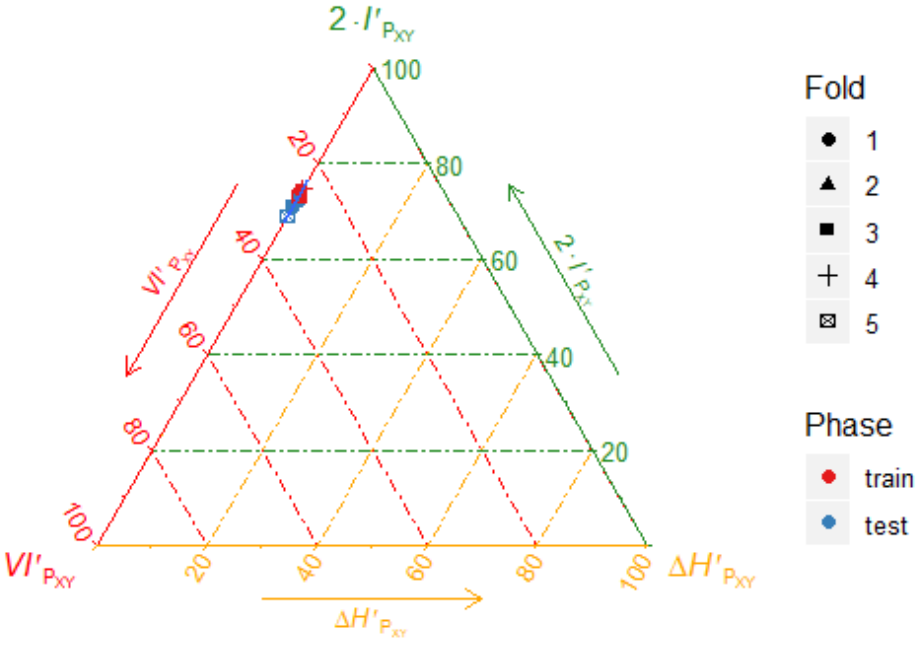


Fig. 3.31. Entropy Triangle confidence interval using Autoencoder + Mlp in MNIST

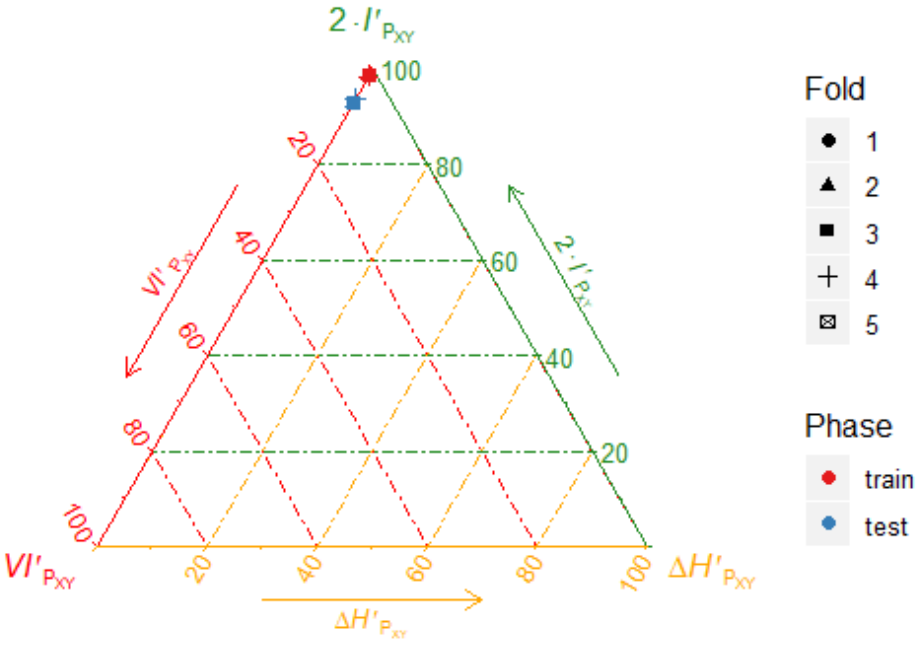


Fig. 3.32. Entropy Triangle using PCA + Mlp in MNIST

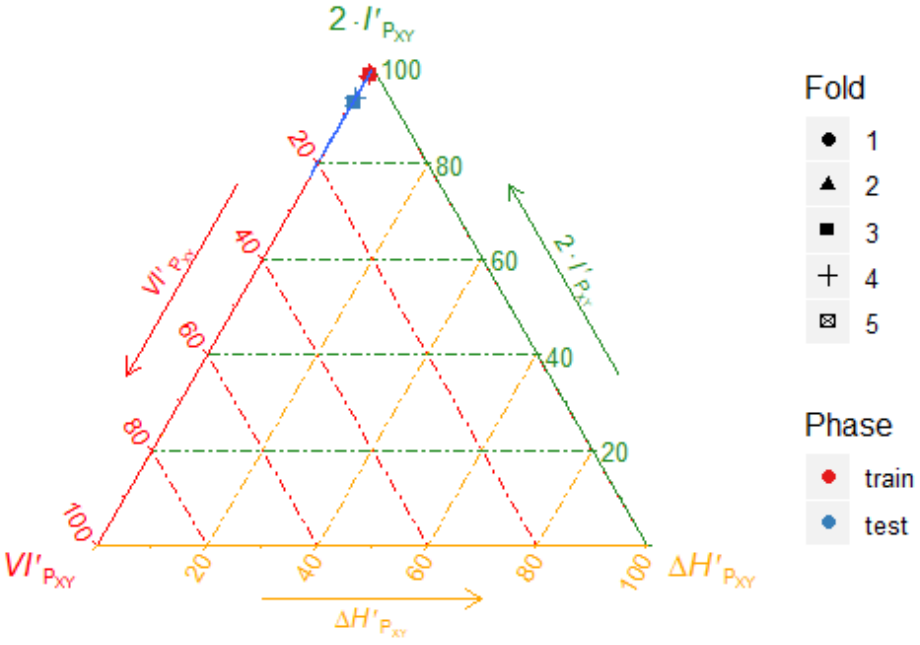


Fig. 3.33. Entropy Triangle confidence interval using PCA + Mlp in MNIST

After plotting both Figure 3.30 and Figure 3.32 we can clearly see that the PCA has proven to be the best choice to classify MNIST. Its training folds have placed in the tip of the ET, and although the test folds have performed worst than them, they are still placed

highly. On the other hand, the MLP classifier on the Autoencoder has good informational characteristic, but it lacks the predictive power of the PCA.

The Figures representing the confidence intervals in this dataset don't provide a lot of information about them. Practically both of them are moving along the $VI'_{P_{XY}}$ line, but regarding delta of the entropies to be constant and the Mutual Information as variable.

3.7.4. Results discussion

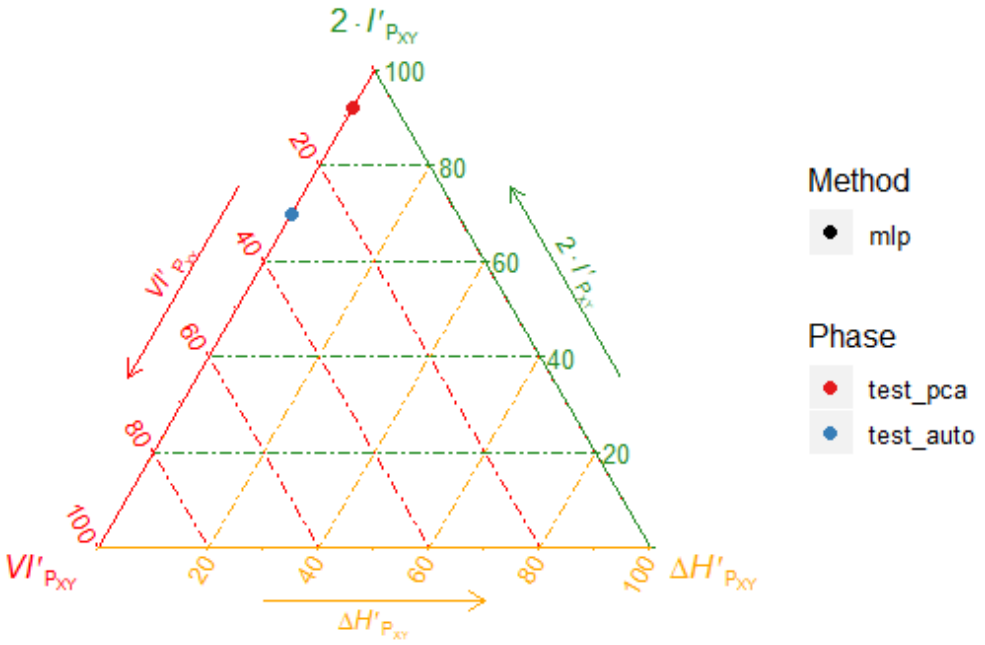


Fig. 3.34. Entropy Triangle in MNIST with the total test fold value

By looking at Figure 3.34 it is confirmed that our previous affirmations about MNIST were correct. The PCA has outperformed the Autoencoder and the test fold has placed the highest out of all the experiment performed in this Bachelor Thesis.

We knew from our investigation before the experiments that great results on the classification of MNIST had been achieved throughout the years, and it has proven to be the most reliable source for the prediction of values to us. It can be due to the fact that it has the greatest number of observations, and despite of its size it can allow for structures such as the MLP to get plenty of data to be trained with and thus get better results at predicting the test folds.

4. CONCLUSION

4.1. Further Work

The possible improvements that can be done to the experiments would be to test the Entropy Triangle on more datasets to further validate our results. We could do so by using some other famous datasets, such as Wine or Boston Housing. It could be interesting to test different transformations such as ICA, but using other Autoencoder structures would be the ideal way to test the ITL. As stated, there is plethora of work available performed with different types of Autoencoder, which could strengthen the validation of the Entropy Triangle by providing more information about how the compression of data is done by them.

The code for the experiments was written using R, so an improvement that could be made would be to translate the vignettes into another programming language, preferably Python since it is one of the preferred tools to program in the data science field. It is also free and available worldwide.

4.2. Conclusion

The aim of this Bachelor's Thesis was to offer reliable information about a tool available for free in the Internet and how it could be used in different environments to test the reliability of experiments. This tool is easy to use and by following the processes mentioned throughout the thesis they would be easy to mimic.

After the problem presented has been tackled, the solution presented was proven viable by using multiple examples and analyzing the results regarded from them. It was also seen that different features of the datasets (such as balancing) affect the classification of their labels. In addition to that, it was also demonstrated that the accuracy paradox is indeed an issue in the data analytics field.

The results presented were consistent with the assertions made before starting them:

- The Iris dataset resulted in being an easy dataset to analyze and the balancing of its

observations allowed the classifiers to overall perform good.

- Ionosphere was heavily unbalanced, so the wrong predictions and thus the bad results found on the ET were expected
- MNIST performed the best when using the ET, and PCA was also proven to outperform the Autoencoder in under specific circumstances.

Improvements such as adding tests and experiments as well as translating the code into another language were also mentioned as a possible way to strengthen the findings in this Bachelor's Thesis.

To sum up, the objectives mentioned at the beginning of the thesis were met, and reliable experiments and information was introduced in order to help researchers around the world improve their methods and, by doing so, expand our knowledge.

BIBLIOGRAPHY

[1] X. Jin, B. Wah, X. Cheng, and Y. Wang, "Significance and challenges of big data research", *Big Data Research*, vol. 2, Feb. 2015. DOI: 10.1016/j.bdr.2015.01.006.

[2] N. Tishby, F. Pereira, and W. Bialek, "The information bottleneck method", *Proceedings of the 37th Allerton Conference on Communication, Control and Computation*, vol. 49, Jul. 2001.

[3] Accessed: 2019-09-21. [Online]. Available: https://www.indeed.es/salaries/Programador/a-junior-Salaries.

[4] T. Mills, *Big data is changing the way people live their lives*. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2018/05/16/big-data-is-changing-the-way-people-live-their-lives/#6444ffc53ce6.

[5] *Big data takes on cancer*. [Online]. Available: https://www.nature.com/articles/d42473-019-00035-5.

[6] s. Nagarajan, S. Gokulakrishnan, K. Ramana, and K. Charugundla, "Application of big data systems to airline management", *International Journal of Latest Technology in Engineering, Management and Applied Science (IJLTEMAS)*, vol. VI, pp. 129–132, Dec. 2017.

[7] European Parliament, *Council regulation (EU) no 679/2016*, 2016. [Online]. Available: https://eur-lex.europa.eu/eli/reg/2016/679/oj#d1e40-1-1.

[8] J. Principe and J. Iii, "Information-theoretic learning", *Advances in unsupervised adaptive filtering*, Sep. 2000.

[9] E. Santana, M. Emigh, and J. C. Principe, "Information theoretic-learning auto-encoder", *2016 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2016. DOI: 10.1109/ijcnn.2016.7727620. [Online]. Available: http://dx.doi.org/10.1109/ijcnn.2016.7727620.

[10] "Understanding autoencoders with information theoretic concepts", *Neural Networks*, vol. 117, Sep. 2019. DOI: 10.1016/j.neunet.2019.05.003. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2019.05.003.

[11] F. J. J. Valverde-Albacete and C. Peláez-Moreno, "100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox", *PLOS ONE*, pp. 1–10, Jan. 2014. DOI: 10.1371/journal.pone.0084217.

[12] U. Gandhi and G. Baskar, "A review on classification algorithms of medical diagnostics", *Research Journal of Pharmaceutical, Biological and Chemical Sciences*, Apr. 2017.

[13] S. Balaban, "Deep learning and face recognition: The state of the art", *Biometric and Surveillance Technology for Human and Activity Identification XII*, I. A. Kakadiaris, A. Kumar, and W. J. Scheirer, Eds., May 2015. DOI: 10.1117/12.2181526. [Online]. Available: http://dx.doi.org/10.1117/12.2181526.

[14] Francisco J. Valverde-Albacete and C. Peláez-Moreno, "Two information-theoretic tools to assess the performance of multi-class classifiers", *Pattern Recognition Letters*, vol. 31, no. 12, pp. 1665–1671, 2010.

[15] F. Valverde-Albacete and C. Peláez-Moreno, "Assessing Information Transmission in Data Transformations with the Channel Multivariate Entropy Triangle", *Entropy*, vol. 20, no. 7, pp. 498–20, Jul. 2018. DOI: https://doi.org/10.3390/e20070498. [Online]. Available: https://www.mdpi.com/1099-4300/20/7/498.

[16] R. A. Fisher, "The use of multiple measurements in taxonomic problems", *Annals of Eugenics*, vol. 7, no. 7, pp. 179–188, 1936.

[17] R. K. Mohapatra, B. Majhi, and S. K. Jena, "Classification performance analysis of mnist dataset utilizing a multi-resolution technique", pp. 1–5, Dec. 2015. DOI: 10.1109/CCCS.2015.7374136.

[18] MNIST, *Mnist — Wikipedia, the free encyclopedia*, [Online; accessed 15-September-2019], 2019. [Online]. Available: https://en.wikipedia.org/wiki/MNIST_database.

[19] V. G. Sigillito, S. Wing, L. V. Hutton, and K. Baker, "Classification of radar returns from the ionosphere using neural networks", 1989. [Online]. Available: https://www.semanticscholar.org/paper/Classification-of-radar-returns-from-the-ionosphere-Sigillito-Wing/e0d2de05caacdfa8073b2b4f77c5e72cb2449b81.

[20] K. P. Murphy, *Machine Learning. A Probabilistic Perspective*. MIT Press, 2012.

[21] A. Zyarah, A. Ramesh, C. Merkel, and D. Kudithipudi, "Optimized hardware framework of mlp with random hidden layers for classification applications", May 2016, p. 985 007. DOI: 10.1117/12.2225498.

[22] J. W. Tukey, "We need both exploratory and confirmatory", *The American Statistician*, vol. 34, no. 1, pp. 23–25, 1980.