uc3m | Universidad Carlos III de Madrid

Bachelor's Degree in Telecommunication Engineering

Academic Year 2018-2019

Bachelor Thesis

Entropy Triangles:

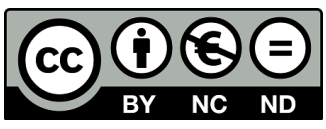
An application to measuring the transmission of entropy in Autoencoders

---

Adrian Manuel de Luis García

Tutor: Francisco José Valverde Albacete

Leganés, 10/23/2019



SUMMARY

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

Keywords:

DEDICATION

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

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 widespread 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 optimise them. It is just assumed that these methods are powerful enough to find a suitable answer.

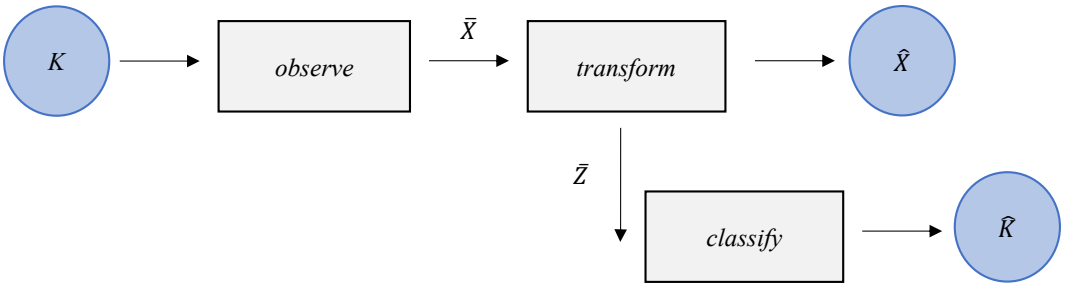
An instance of this model can be found in Figure 1.1.a, where each block represents a different function inside an end-to-end informational model. We can only take measures outside each of the blocks and the contents or the methods being used inside are outside of the scope of the task. By not being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

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 **refToInfBottleNeck**.

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

[1] (Quote article to reinforce my position on this matter? FVA: YES!)

(a) Conceptual representation of a classification process mimicking a communication scheme of boxes.



(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).

- 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 $\overline{X}$ (X con gorro) 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.

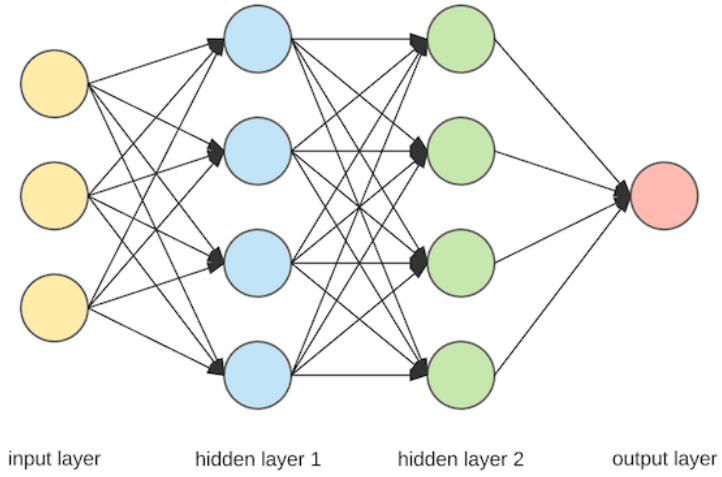
Note that Figure **??** follows a similar model to that of Figure **??**, but its transformation block is used to access the inside content of it rather than to provide a typical representation of an end-to-end transformation scheme. Although both of them typify a MIMO (Multiple Input Multiple Output) block, the Autoencoder is essentially an unsupervised transformation method, while the transformation block can be either composed by supervised or unsupervised task. The inner workings and different features of the Autoencoder will be explained on Section 2.1.

Next is a review r the concepts to be used on this Final Degree Project and the rest of methods to use on it.

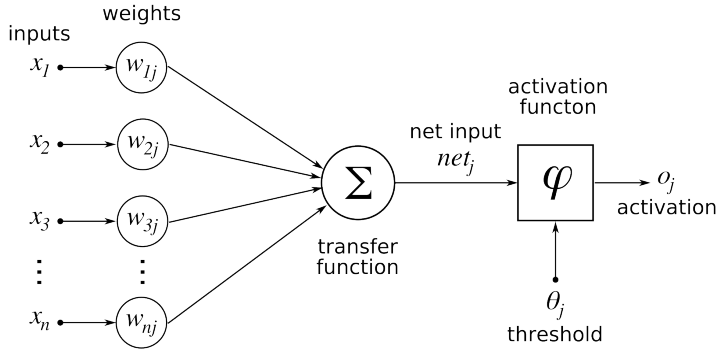
2. BASIC PRINCIPLES

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 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 Figure2a, 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 and 2.1. 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).

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

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.

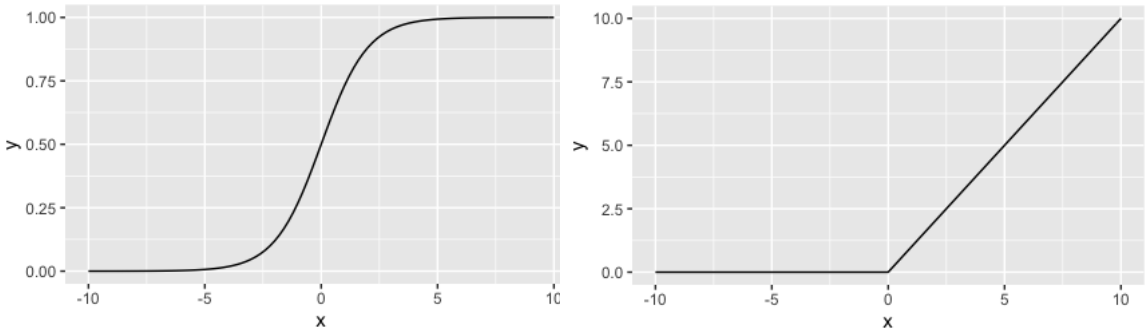


Fig. 2.2. The left graph represents a Sigmoid function, while on the right there is a Relu function

As it can be seen on Figure2.2 , the Relu function has a relatively simple representation, which makes it less computationally expensive when compared with the Sigmoid. The Sigmoid function is also very useful since it 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 neurones inside of the network will start to propagate the information through it in a non-linear manner in order to try to achieve a given stablished 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, visualisation,....etc.

2.2. Information Bottleneck Principle

The architecture of the Autoencoder is based upon the IB, a method proposed in [paper which mentions it] whose principle relies on extracting the relevant information that an input random variable 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).

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), or simply just the mapping of X that retains I(X;Y).

During the learning process, and as depicted in figure (DNN self-explanatory figure), 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 out network: the information not processed on the last immediate layer is essentially lost. This is the main reason why every layer should attempt to maximise I(Y;hi) while trying to minimise the I (hi-1;hi). Here, it is important to consider then that we want to reduce the length of the layer to the minimum possible without affecting the 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 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 on both previous sections (the two sections appearing before this one, when i figure the final numbers), the autoencoder comes as a mixture of them. Figure [] below presents it as a three part structure: A encoder, the middle layer and finally the decoder.

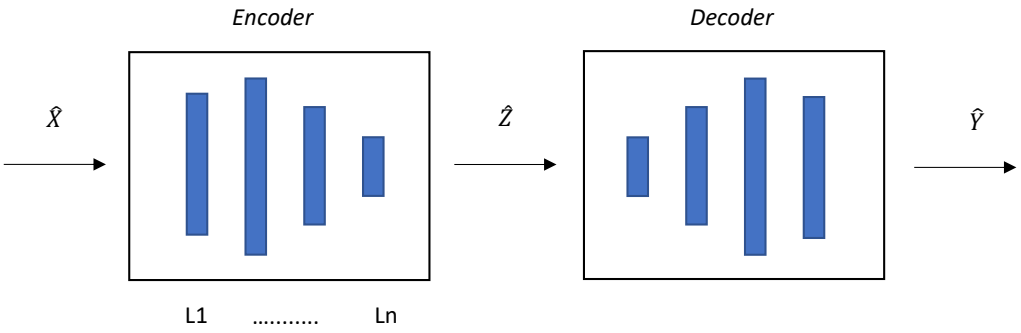


Fig. 2.3. Taking into account the Figure 1.1b, this diagram depicts the inside of the transformation box. Both the Encoder and the Decoder have symmetric shapes, as well as an expansion layer (the second one in the encoder, the previous one in the decoder) whose goal is to separate the qualities of our data as sort of a preparation for the compression.

The theory of the autoencoder has been around since the 1980s (put as a reference the article where it is first presented). As seen in Figure[], the autoencoder's goal 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 optimise 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 variances of them exist, such as the recirculated autoencoders.

In our case, we will be also using an undercomplete autoencoder. Most of the types an autoencoder is employed, 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 on the input data X in the output on the encoder, which for the purpose of this report we will denominate as Z. The learning process, as described in (paper where i got the equation from) tries to minimise the given equation:

$$L(x, g(f(x))) \tag{2.2}$$

The L function is a loss function, which penalises $g(f(x))$ 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 optimised autoencoder will struggle to perform this task, specially when introducing very big sets data as the input.

Taking into account this description, the general rule to design it is just by using trial and error. As we will see on Section3, 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 report, we will not only taking into account those parameters, but we will also be using a tool to ensure that it transfering the higher quantity of information possible: The entropy triangle.

2.3.2. The Entropy Triangle

Introduction

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.

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 to the face recognition task. That is why it is specially important to correctly address the scope of your task and how accurately does your model predict in your pre-defined scenarios.

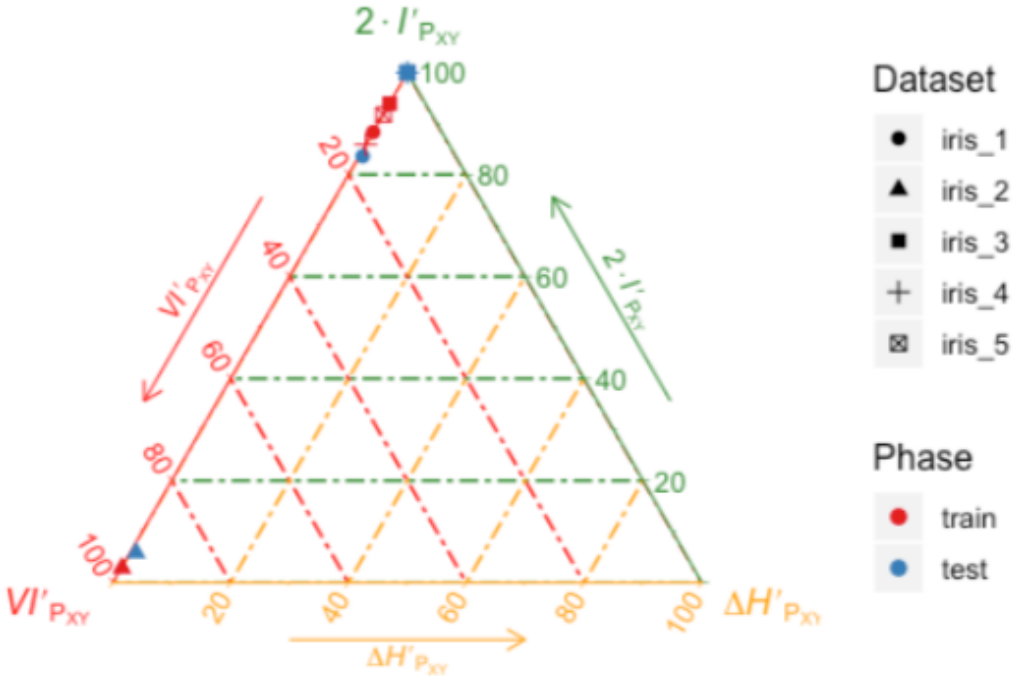


Fig. 2.4. Example of use of a typical Entropy Triangle. In this case, a Knn classification method is being tested in the triangle.

The real question comes when you have your results and you try to wrap your mind around them. In [paper donde se discute ese tema] a very important reality is shown: the accuracy performance criterion is a very intuitive but ineffective 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 that other with a higher accuracy. The reason being 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 to.

Once it is realised that the methods previously employed can lead to disingenuous

results, a new realisation arrises: there is a need for a better measure of the classifier. As seen in, 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 functionalities.

Architecture

The Entropy Triangle was introduced in ... as a way of solving the problems introduced in the previous section. 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 Figure1.1 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 we can see a classical information-diagram ... which simply show 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:

$$MI_{P_{XY}} = H_{P_X * P_Y} - H_{P_{XY}} \tag{2.3}$$

and,

$$MI_{P_{XY}} = H_{P_X} - H_{P_{X|Y}} \tag{2.4}$$

The Virtual Information or 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:

$$VI_{P_{XY}} = H_{P_{Y|X}} + H_{P_{X|Y}} \tag{2.5}$$

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:

$$H_{U_X * U_Y} = \Delta H_{P_X * P_Y} + 2 * MI_{P_{XY}} + VI_{P_{XY}} \tag{2.6}$$

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 equation2.6 we will normalise 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:

$$1 = \Delta' H_{P_X * P_Y} + 2 * MI'_{P_{XY}} + VI'_{P_{XY}} \tag{2.7}$$

Which also will wield the following equation:

$$0 \leq \Delta' H_{P_X * P_Y}, MI'_{P_{XY}}, VI'_{P_{XY}} \leq 1 \tag{2.8}$$

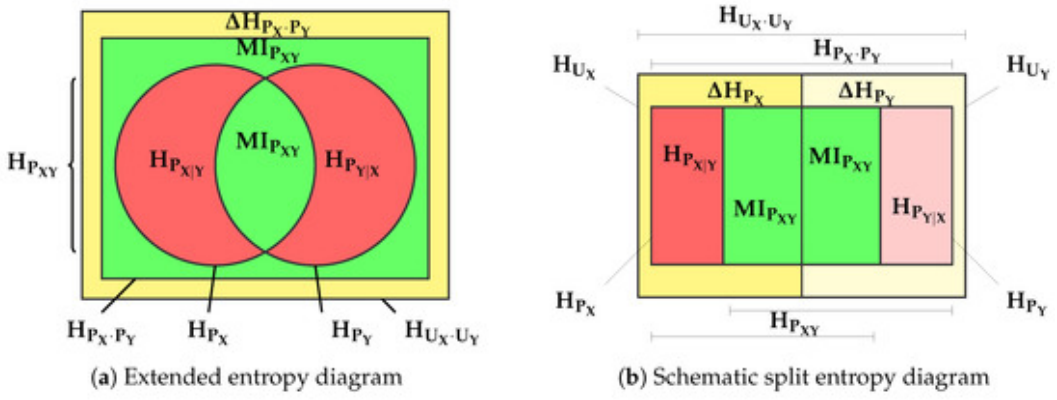


Fig. 2.5. Diagram representing an ET applied to a bivariate distribution.

By applying both equations 2.7 and 2.8 we will get a point in the normalised space $\Delta' H_{P_X * P_Y} x 2 * MI'_{P_{XY}} x VI'_{P_{XY}}$. Each P_{XY} can be characterised 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 bivariate 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 2.6 to obtain new representations of the split balance equations with respect to X and Y,

$$H_{U_X} = \Delta H_{P_X} + MI_{P_{XY}} + H_{P_{X|Y}} -> 1 = \Delta H'_{P_X} + MI'_{P_{XY}} + H'_{P_{X|Y}} \tag{2.9}$$

and,

$$H_{U_Y} = \Delta H_{P_Y} + MI_{P_{XY}} + H_{P_{Y|X}} -> 1 = \Delta H'_{P_Y} + MI'_{P_{XY}} + H'_{P_{Y|X}} \tag{2.10}$$

Both equations are normalised by using both H_{U_X} and H_{U_Y} respectibly. 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.

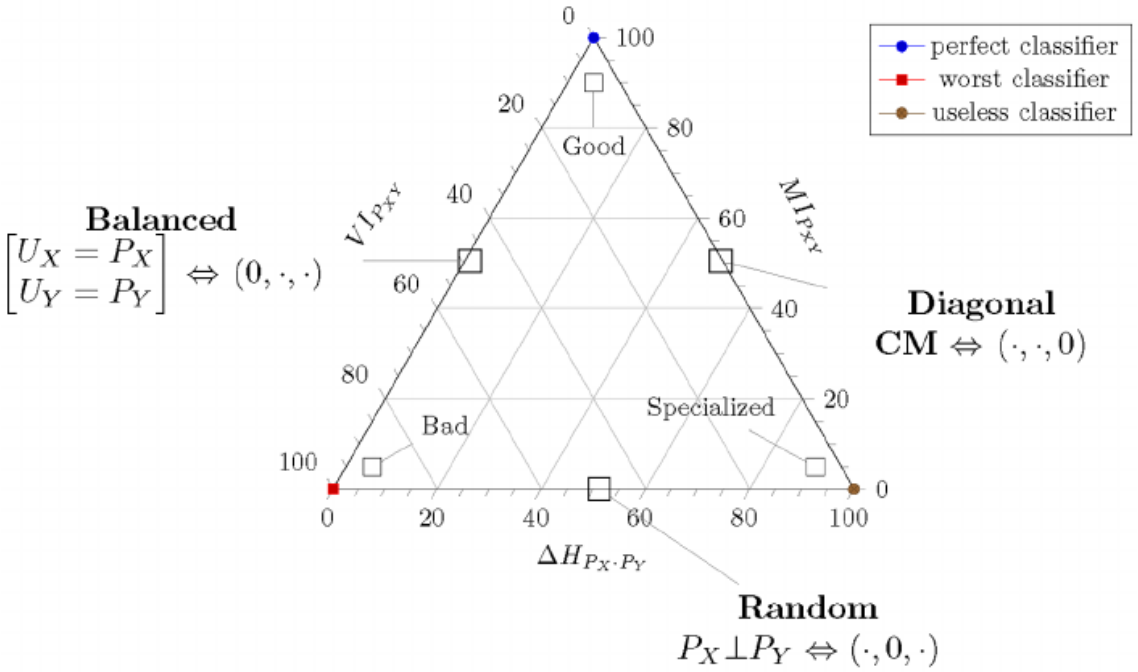


Fig. 2.6. Schematic of an Entropy Triangle. The labels are placed close to the side of the triangle that is related to the feature mentioned in it.

Most of the equations included here can be found in [] 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 use.

The triangle itself works as a great tool to characterise the performance of your model. The position of the coordinates assesses the quality of your classifier. It can be appreciated from Figure 2.6 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 specialised classifier which tries to work with very unbalanced data.

One example of use of the Entropy Triangle which appears on Section3 together with the results of the experiments would be to use the true labels K and the predicted labels $\hat{K}$ to generate a confusion matrix. Information to be fed the ET can then be calculated with it and thus provide the diagram with enough materials to be created. The tool can then be used for exploratory analysis and help provide visual data to assess the feasibility of the task and its effectiveness.

2.4. Principal Component Analysis

2.4.1. Description

The Principal Component Analysis in 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 of vector which are uncorrelated. Due to the nature of the transformation, PCA can be affected by any type of scaling of the original dataset.

PCA is used in the data analysis field for exploratory data analysis or to make predictive models. It can be used to visualise 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. Both of them are used to describe the variability of the data and reduce the dimensionality of the data, but aim to use different techniques to reach that goal.

- It also resembles canonical correlation analysis. 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.

2.4.2. PCA and Information Theory

Our goal when using PCA in this project is to try to give other examples of data transformation to compare the results of the Autoencoder with another widely used tool in the data science field. At the same time, we want the Information theory to be implicit on both methods used so that the outcome can be correctly compared when using the Entropy Triangle.

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

$$x = s + n \tag{2.11}$$

Where x represents the vector being the sum of the desired information-bearing signal s and a noise signal n. If we use this equation, it can be assumed that our vector can be dimensionally reduced.

For this model to hold its truth, our signal n has to be Gaussian with a covariance matrix proportional ti the identity matrix. This fact allows us to maximise the mutual information between the dimensionally reduced output y and our s signal, only if we consider that the same assumptions made for n also apply for s.

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

$$I(x; s) - I(y; s) \tag{2.12}$$

If our noise is dependant, the PCA losses its informational properties and thus we

cannot use the previous representations. On the other hand, if our noise is more Gaussian that our bearing signal *s*, our PCA representation will be optimal.

3. STATE OF THE ART

3.1. Information Theorethic Learning on Autoencoders

The Information Theorethic Learning concept was introduced in **Principe_2000** 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 manipulate information. The name ITL is a natural transition from the aforementioned proccess, as the machine learning function had to be independent from the informational transition of data.

The Autoencoder was first proposed to test the concepts of ITL on **Santana_2016**. Using similar theoretic descriptors as the Entropy Triangle, like 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.

$$cost = L(x, \overline{X}) + \lambda \times R(E, P) \tag{3.1}$$

On Equation 3.1 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.

On **Yu_2019** Principe and Yu 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(an structure also proposed in this report) they use widespread available materials and datasets to evaluate the capacity of a stacked Autoencoder (as we do on our experiments) to demonstrate the ITL.

4. DEVELOPMENT

4.1. Exploratory Analysis

Before diving into the application of the Autocoder, we firstly 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 assesments 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 in our interest to prioritise them according to a series of qualities. They needed to:

- Have different lengths, ranging from easily handable datasets (Iris) to more complex and computationaly harder (MNIST).
- Different classification taks, from binary to categorical.
- Clearly differentiable balancing in the datasets, which will help us understand the potential of our tool.

The following section will help understanding these points and its implications on our task. To do so, a brief description of them and its role on achieving our desired results will be tackled. Descriptions of each dataset will include statistics on their distributions as well as why they were chosen as a viable candidate to fulfill 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 neccesary 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 a light in the difficulty to adapt our scripts and architectures when moving from one dataset to another one.

4.1.1. Iris

The Iris flower dataset is a multivariate data set published in the 1936 paper (The use of multiple measurements in taxonomic problems) by Ronald Fisher. The data itself consists of 50 samples from each of the three species of Iris: Iris setosa, Iris virginica and Iris versicolor. Each of them had four of their characteristics tested: the length and the width of the sepals and petals in centimetres.

Based on the distribution of the measurements regarded 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.

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 4.1. R SUMMARY METHOD ON IRIS.

The table of top contains a summary of the most important statistical features of iris, and by looking at it we can already see that the data is very balanced on every feature as the distance between the Minimun and the Maximun divided by two is approximately the Mean, which also has a close value to the Median. To us, this means that the data is centerred 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.

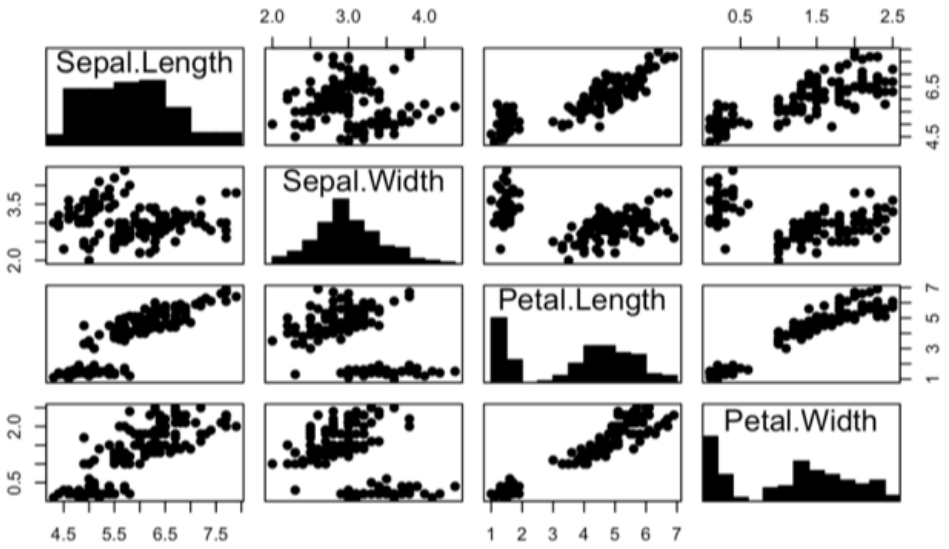


Fig. 4.1. Using the pairs function summary

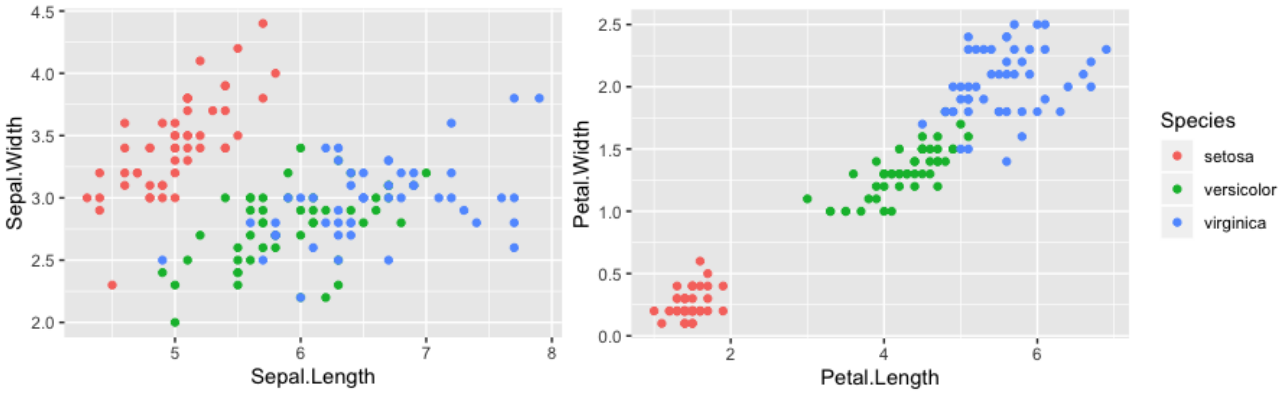


Fig. 4.2. Plotting of two different features of Iris with respect to the class label. As it can be seen here, depending of how you organise your data you can get more efficient classifiers and clusters.

Taking a look at Figure 5.3, The histogram on the middle is showing some higher values in different ranges on our data. If we hadn't observed the summary values on the Table 4.1 , we could have struggled coming up with the general idea behind the dataset. But sometimes histograms are a little bit more troubling to read into, so plotting Figure 4.2 can help understanding 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 distinguinshable when using Petal Width and Petal Length rather than when using Sepal Width and Sepal Length.

4.1.2. MNIST

The MNIST dataset is a large database of handwritten digits widely used in the data science field both for training and testing. 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 normalised to fit into 28x28 pixel images. Every image has a value from 0 to 255 ,which defines the shape and color of the number in the image. Then, the y class contains vectors with a character which can be related to its corresponding pixelised image by a classification method.

The database contains 60000 training images and 10000 testing images for X and Y. Many different papers (name one) have tried to achieve the lowest error rate on it. Convolutional neural networks manage 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 0.8 too.

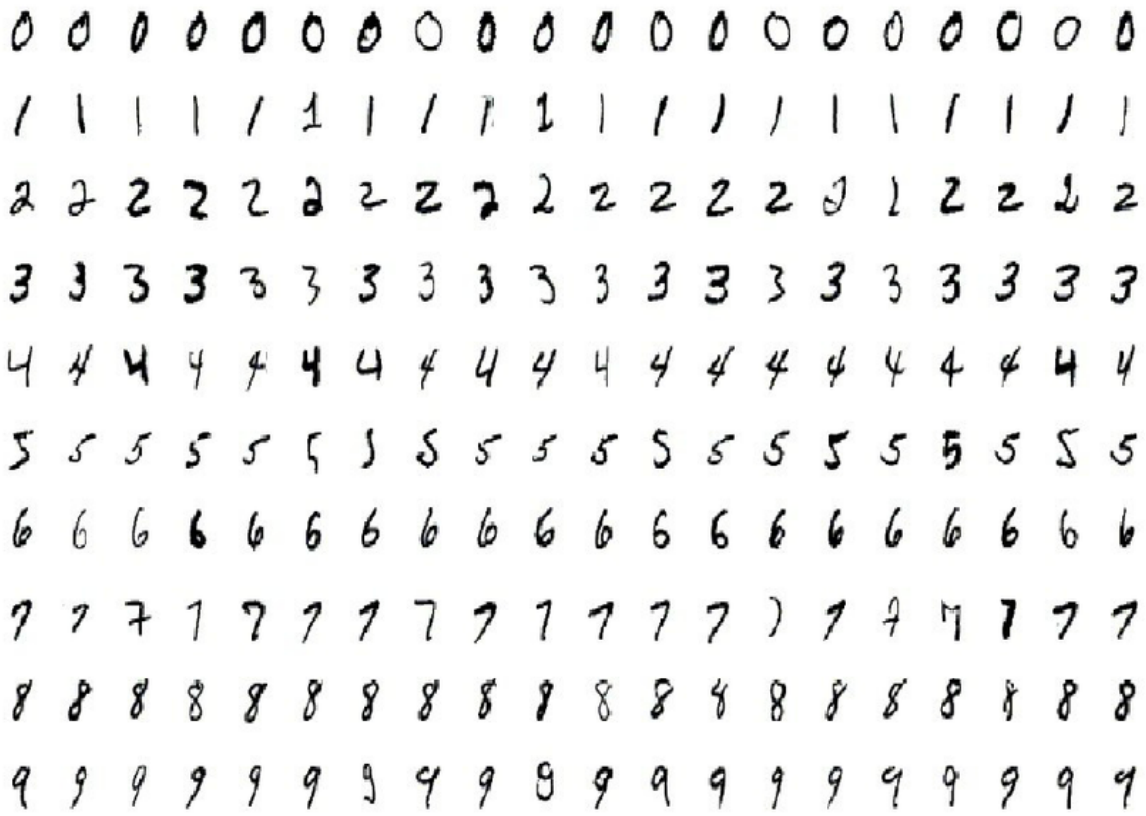


Fig. 4.3. Some examples of the images included in the MNIST dataset

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 4.2. R SUMMARY METHOD ON MNIST.

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 4.2, we just will know the range of values of every image (already discovered in the definition on the dataset) and the Mean, which only states that out training pixels tend to be white rather than black. In this case, since all of our bits are white or black (255 or 1 in decimal representation), 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.

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 (located in the tensorflow package) 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 its expected outcome in terms of error rates and accuracy. We are not trying to improve their results, we are just mimicking their processes and applying some of their methods so that we can achieve similar results when applying our tools.

4.1.3. Ionosphere

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. The antenna had a phased 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 labelled as "bad", and in any other case we would be labelling it as "good". It can be found on the R package mlbench.

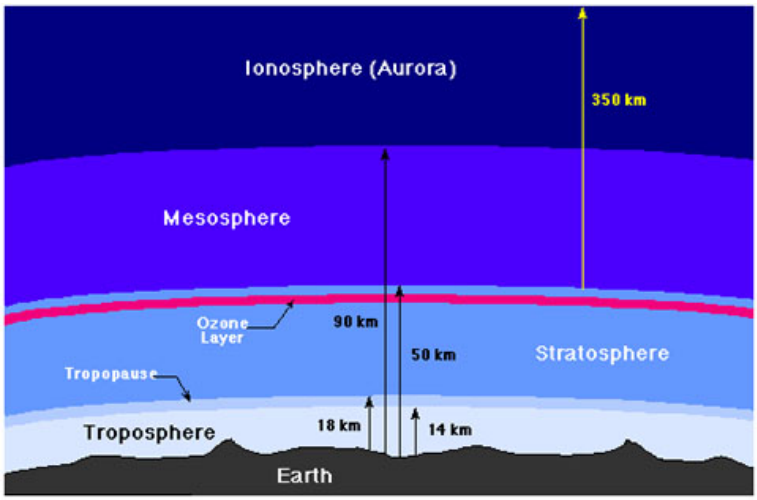


Fig. 4.4. The Ionosphere is one of the main layers composing Earth's atmosphere

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 at [Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B. (1989). Classification of radar returns from the ionosphere using neural networks. Johns Hopkins

APL Technical Digest, 10, 262-266.] have found very high acurracies when using non-linear perceptrons on this dataset, but to do so they had to use the quantities of "good" and "bad" observations, which would not fit our needs of information. 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 our tool.

4.2. The classifiers

4.2.1. Knn

Knn is a non-parametric supervised classification method used for estimating the density function. As an unsupervised 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. This theory 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.

Although "the majority voting" classification method has some positive advantages, it also comes with some drawbacks too. In case where the class distributions are skewed, we can end up with a class which solely dominates the predictions. The 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.

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

- Try fitting your data through multiple instance 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, realising that a problem will be very lengthy and tedious to solve using a pre-determined method will save you a lot of time .

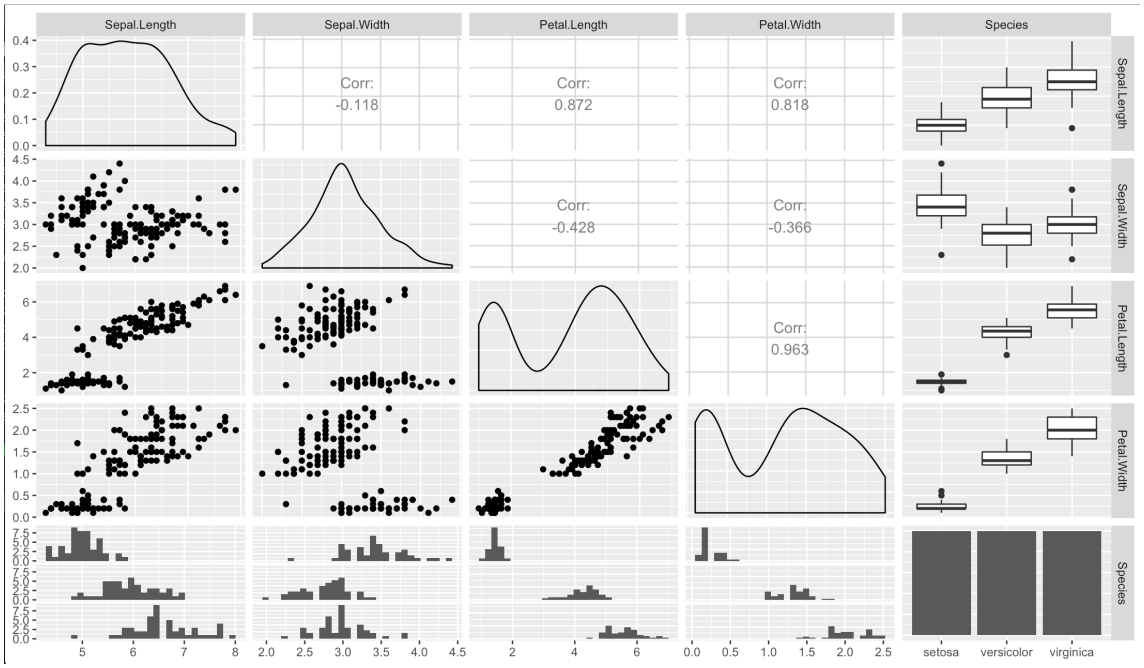


Fig. 4.5. The GGally package also provides useful tools for this types of problems. Here, the correlation between Petal.Length and Petal.Width is proven to be very high, as the plots from Figure 4.2

4.2.2. Multilayer Perceptron

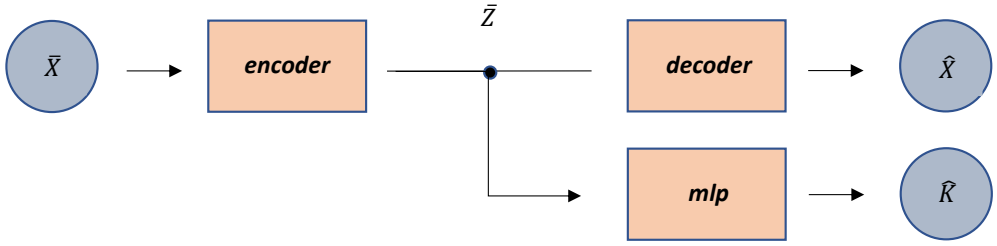
The Multilayer Perceptron(MLP) has a similar architecture to that of the Deep Neural Networks. Both of them have input, output and hidden layers as well as having nodes and backpropagation for training. They even need their layers to be activated, as well as support the most common functions used in the autoencoders. Some experts would even label it as a slightly less computationally powerful DNN.

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: it can distinguish and classify data that is not linearly separable.This property can help us qualify and compare different

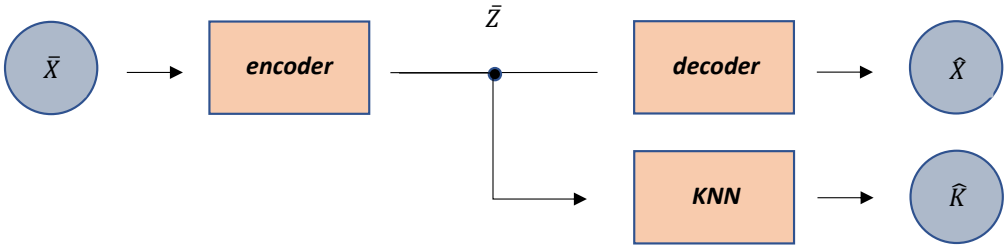
types of classification tasks. And as it is interesting to test multiple algorithms to make the most out of our ET, we can use it together with the rest of the classifiers presented in this Section. It can be specially useful when we pair it with the knn classifier, as we will implement both of them in our project to perform the same tasks. We would expect it to generally perform better on more complex datasets than knn, which suffers when data is more difficult to differentiate.

5. RESULTS

In this section we will be presenting the results regarded from 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, which will help us discern the capabilities of the tools involved. To be able to do so, we have to establish the same structures of tests.



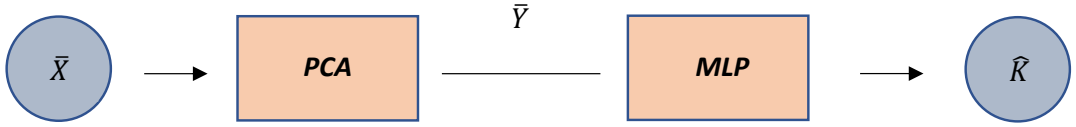
(a) Autoencoder and MLP.



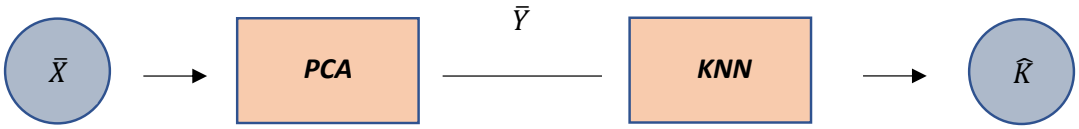
(b) Autoencoder and KNN.

Fig. 5.1. Structure of the Autoencoder and it's classifiers

As seen in Figure 5.1, we will use the representation of our encoded data $\hat{Z}$ as the input for both the KNN and the MLP classifiers.



(a) PCA and MLP.



(b) PCA and KNN.

Fig. 5.2. Structure of the PCA and it's classifiers

If we compare both the structures presented here, we can see in the images in Figure 5.2 that we will be using the output from the PCA as the input of the classifiers, as well as that we will have the same classifiers used in both architectures, with the aim to be able to compare their results.

As we are also concern about the reliability of them too, we will be including another technique to try to validate our results as much as possible: K-Fold validation.

K-Fold is a method which is based on dividing a dataset into K folds, each one of them not sharing the same elements on their observations. We are interested into really asserting how each of the Folds of our datasets performs by fitting them through the structures presented in both Figure 5.1 and 5.2. Once we have each one of them complete the full training process, we will analyse the output plus the summation of the overall results. We expect our Folds to 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.

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 generated at the beginning of the process and the output of the classifier.

5.1. Iris Dataset

5.1.1. 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 normalisation 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 **??** and Table 4.1, some of its features are closer to what it could be referred as a Normal distribution, specially Sepal Length and Sepal Width. Other's, 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:

$$y(\lambda) = \begin{cases} \dfrac{y^{\lambda} - 1}{\lambda}, if\ \lambda \neq 0 \\ \log y, if\ \lambda = 0 \end{cases} \tag{5.1}$$

Where λ is an exponent which 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 5.1 .

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

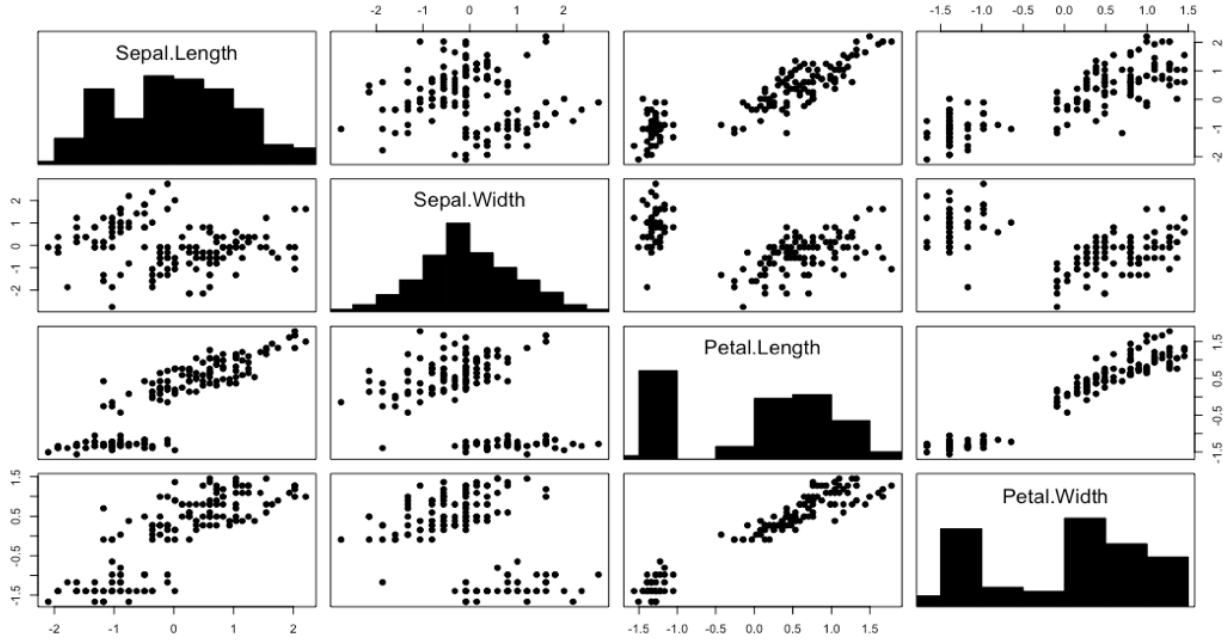


Fig. 5.3. Using the pairs function on Iris Boxcox

Which has an histogram slightly different to the one presented on Figure 5.3, 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 4.1

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 5.1. 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.

5.1.2. 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 Autocoder has the aim to show an example of use. That is why , if we take into account that we only have for variables to compress, we can see that the maximum compression we can achieve is a 25 percent of the original size. But, in order to be sure we have a working architecture, I will be using just a 75 percent compression in the middle layer. On the other hand, our model will have a data expansion of about 400 percent, so that we are able to do a real compression in each of the subsequent layers.

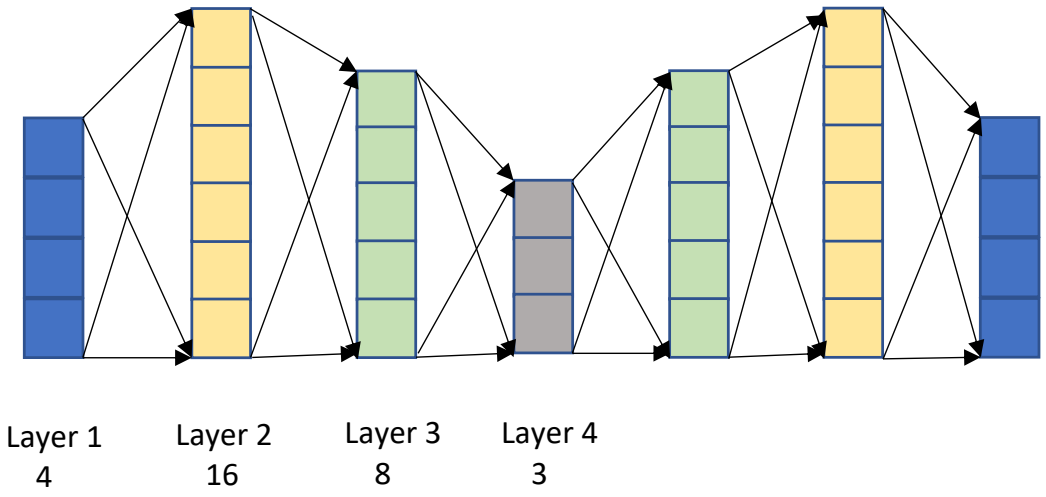


Fig. 5.4. Architecture of the Autoencoder in Iris

On Figure 5.4, 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 finalise 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, to try to maximise the performance and optimisation 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 stablished that, we have a narrower set of possible choices to make. Between all of the available, 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 them, the 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 optimiser for our compilation. Here we decided to narrow our options to chose only between the Adam and the Adadelta options. Finally, it was seen that the Adam optimiser 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.

5.1.3. 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. 5.5. 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, labelled

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 we used to train it on the first place. Once we have calculated that, we will use the confusionMatrix 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.

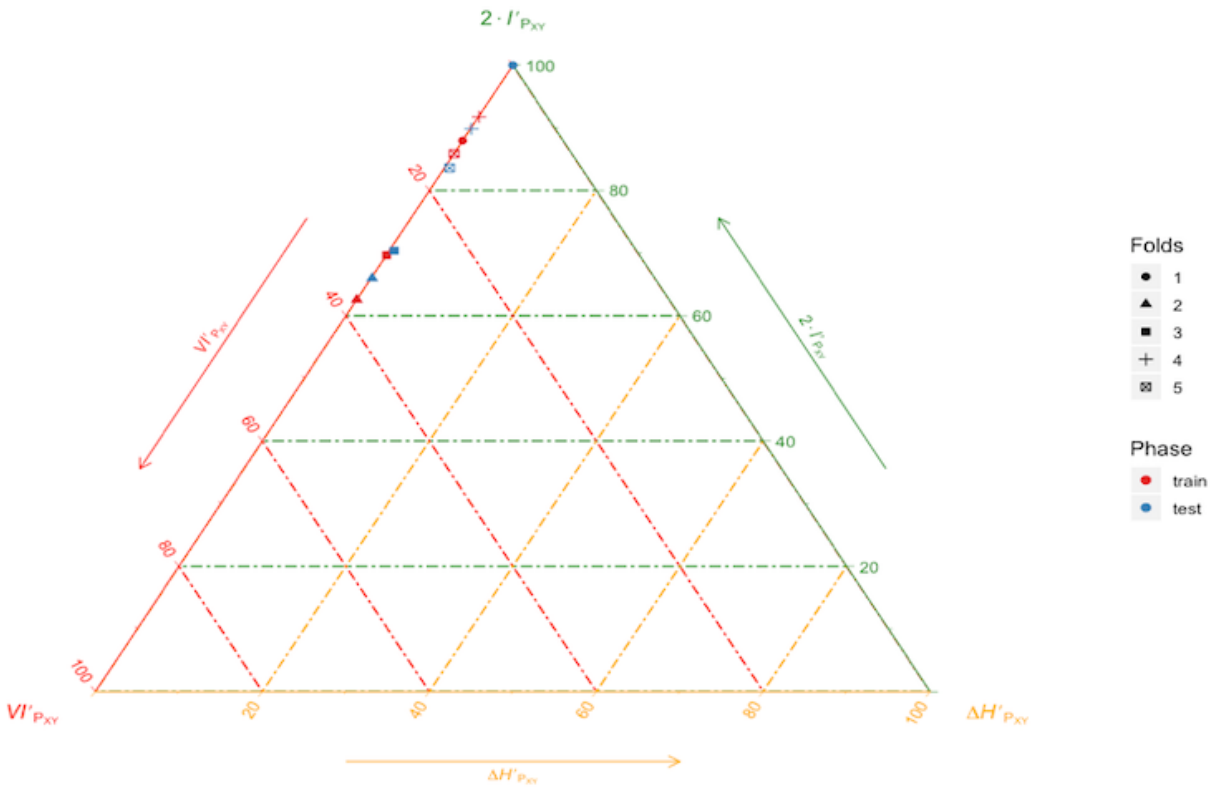


Fig. 5.6. Entropy Triangle in Knn in Iris using the Autoencoder

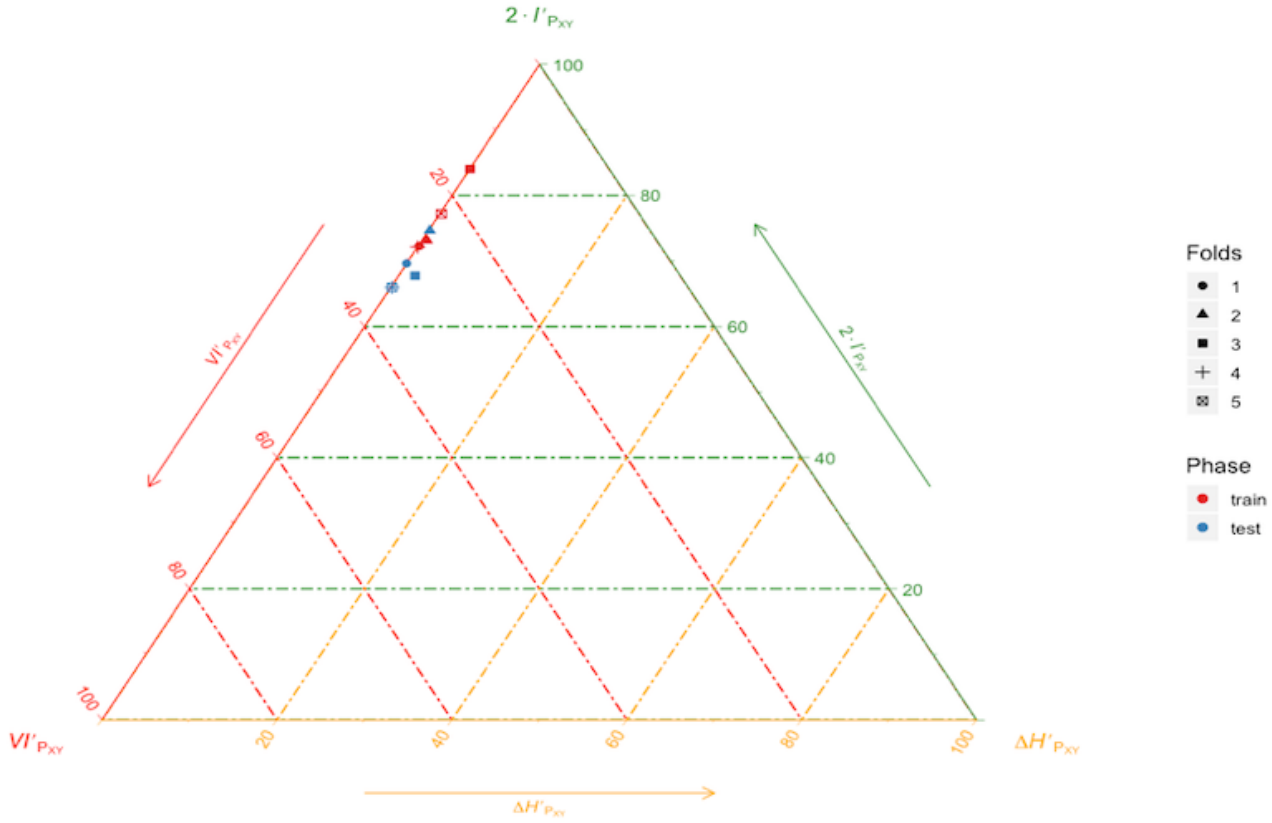


Fig. 5.7. Entropy Triangle in Knn in Iris using the PCA

This method will provide us with metrics that then can be used together with the ggmetern method to get a coordinateS inside de Entropy Triangle, which can be seen on both Figure 5.6 and 5.7. 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 the Knn on the Autoencoder as seen on Figure 5.6, we can see that the coordinates from the training and testing are close to each other spatially, which means that we have done a good job overall at training our model. Some of the folds seem to have performed worse that the other, but that can just be that the other performed extremely well with their predictions. Specially the First fold seems to provide the best results, as we are getting a perfect classification as well as the optimum information flow on our Entropy Triangle. To completely be able to asses it's quality, we have to see the overall values of the total Confusion Matrix of the testing phase.

On the other hand, the PCA on Figure 5.7 has a very centralised amount of values. The majority of the phases from the folds seems to be placed around the same area of the Entropy Triangle. No fold has over-performed the others in a greatly manner, which tells us that the results are consistent all throughout the training, which is not the same case as in the Autoencoder case, where the results are more spread.

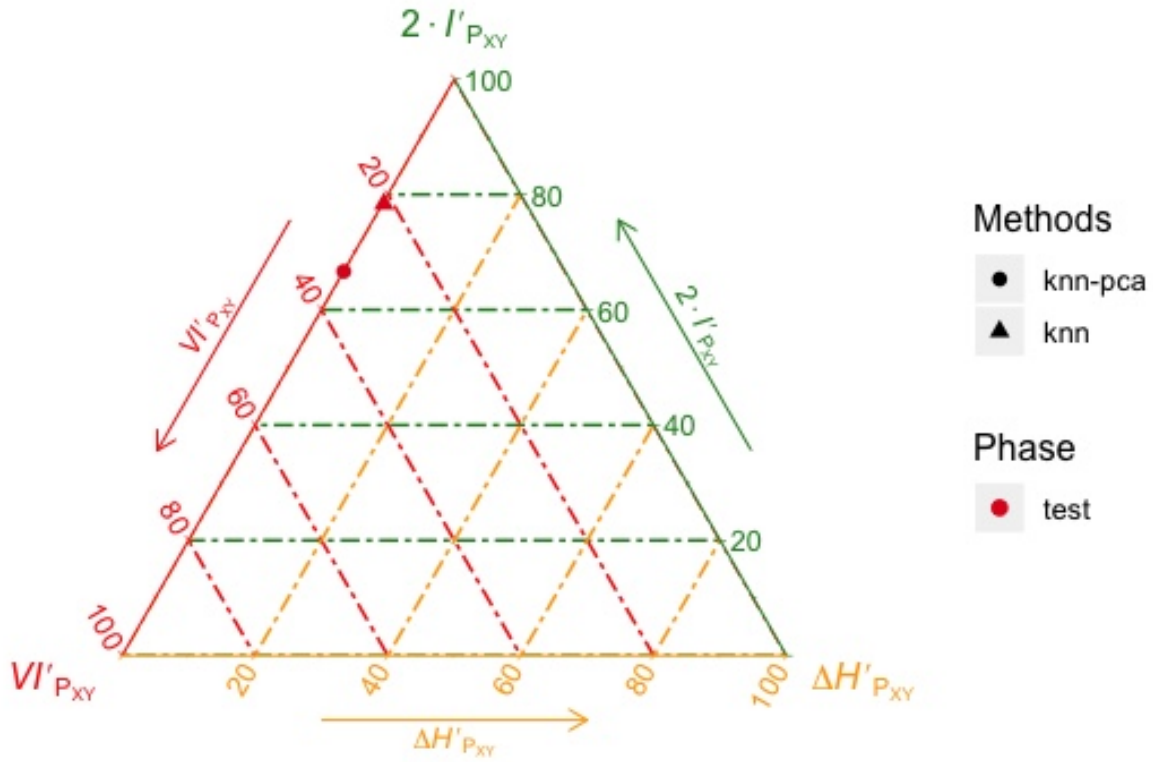


Fig. 5.8. Entropy Triangle in Knn in Iris with the testing results

Although the pca has more consistent and less spread results, we can see in Figure 5.8 that the Autoencoder has done an overall better job at providing information for the knn for the classification. Both of them are placed on the side of Triangle $VI_{P_{XY}}$ that accounts for the transference of information, but the Autoencoder is providing a better representation of the data that is allowing the knn to do a better job overall.

We can therefore conclude that, in the case of the classification using the knn on both the PCA and the Autoencoder, we have found the latter to do a better job than the first one informationally speaking, thus leaving us with the assumption that it is the preferred solution for the Iris dataset.

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.

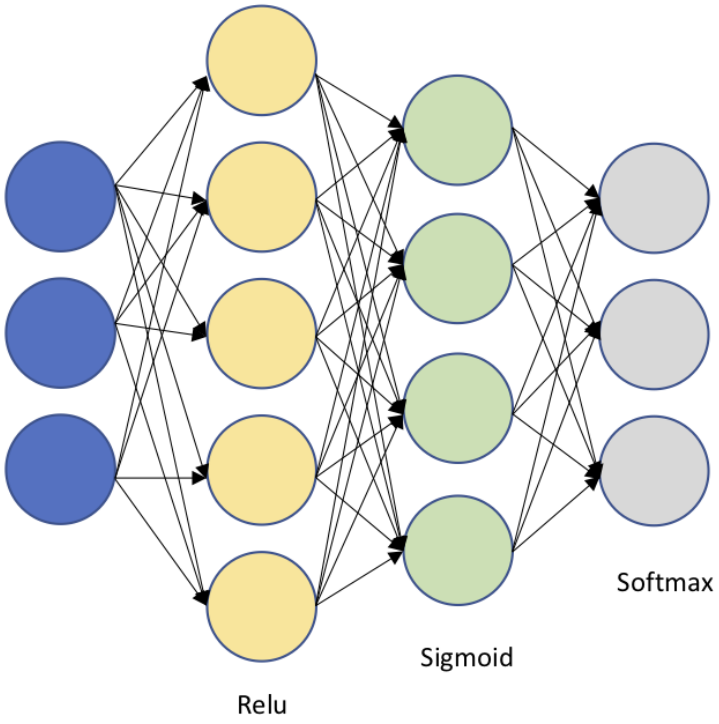


Fig. 5.9. MLP architecture in Iris for the Autoencoder

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.

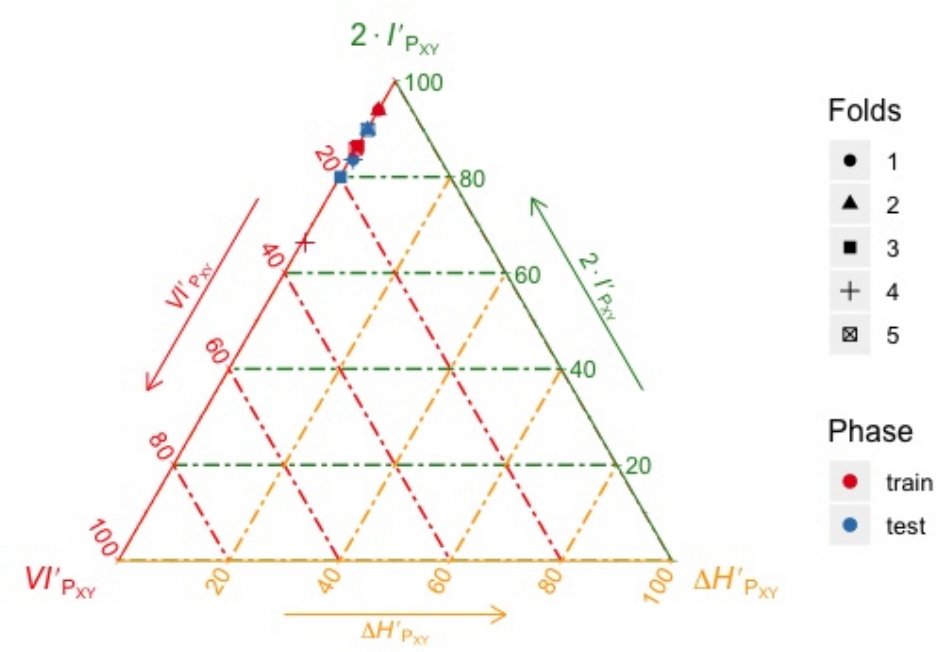


Fig. 5.10. MLP Entropy Triangle in Iris for the Autoencoder

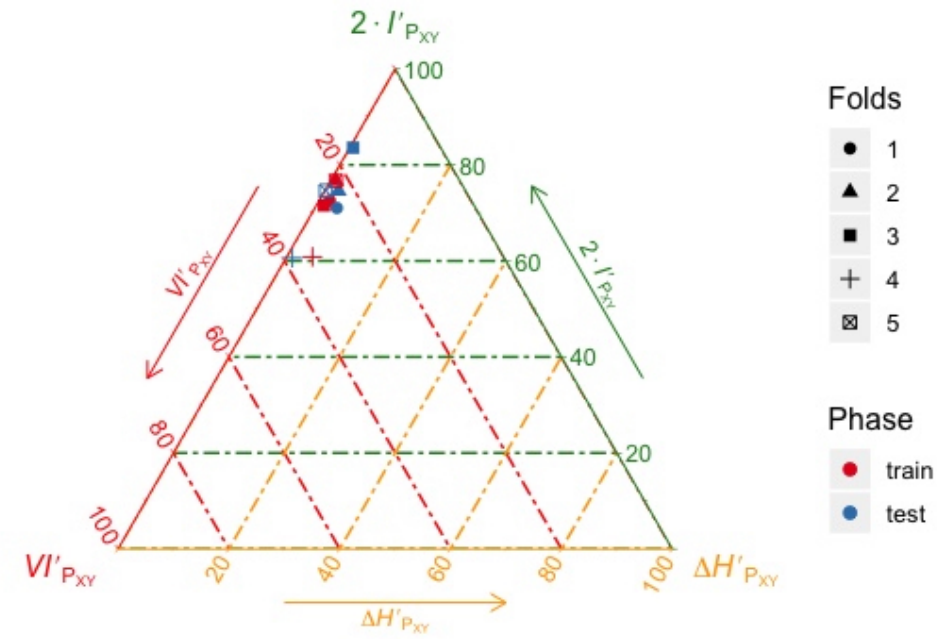


Fig. 5.11. MLP Entropy Triangle in Iris for the Pca

From Figure 5.11 has produced again spatially closed coordinates, which tell us that the folds have been very consistent at training our model. The fourth fold seems to have given slightly worse results, but overall the mlp has generated values that are consistent with our prediction that it would work well with Iris. We can also see that most of the values are grouped on the diagonal of the $VI_{P_{XY}}$, which tells us that our folds have been correctly balanced.

On the other side, we have Figure 5.10 which seems to have slightly better results overall. We can see the same behaviour on the fourth Fold as before, which tells us that the mlp may have struggled on both cases to train that fold. All of the test folds are close spatially and in the previously mentioned $VI_{P_{XY}}$ diagonal as well as placed very highly on the Entropy Triangle, which tells us that the mlp is being able to classify efficiently.

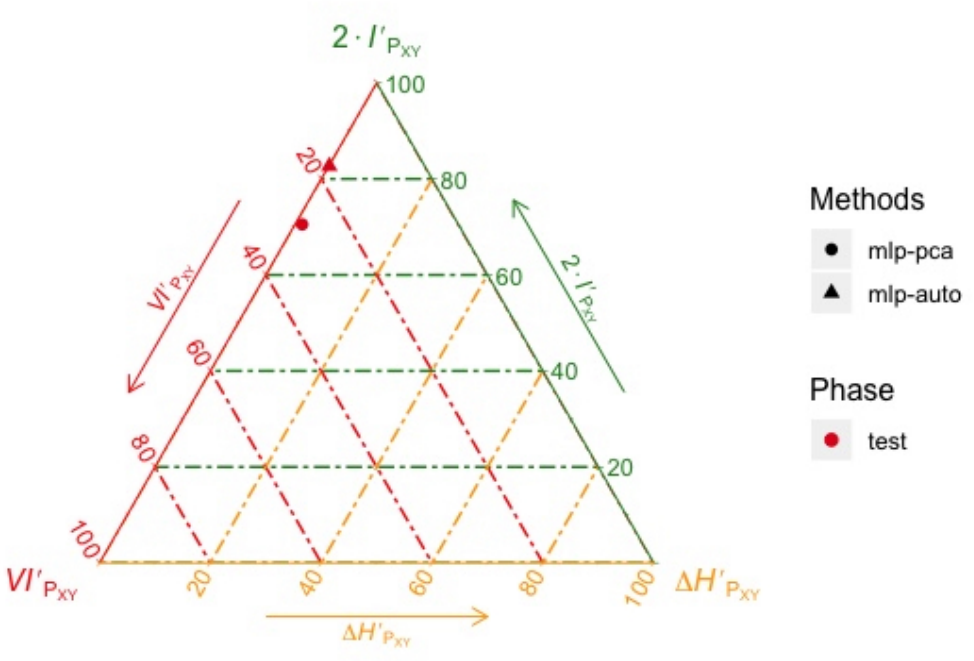


Fig. 5.12. MLP Entropy Triangle total testing result

As in the previous cases, we finally have to look at the overall performance of the mlp. In this case, it can be seen that Figure 5.12 shows a clear advantage of the Autoencoder for the same task. Moreover, if we compare both the results from the Knn and the MLP, we the MLP has the best results for the classification task. On the other hand, we are not seeing a big difference in the performance of the MLP with respect to the KNN, but that may also be because of the simplicity of the Iris dataset.

5.2. Ionosphere Dataset

5.2.1. Data Preparation

The only pre-proccessing done for Ionosphere was explained in the previous section referred to the 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.

5.2.2. 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. The structure that we decided to implement 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 5.2. 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 5.3. IONOSPHERE AUTOENCODER LAYERS OF THE DECODER.

As seen of both Table 5.2 and Table 5.3, 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 behaviour on this experiment.

5.2.3. Knn on PCA and Autoencoder

We are applying the same process mentioned on **??** to plot the ET in Ionosphere.

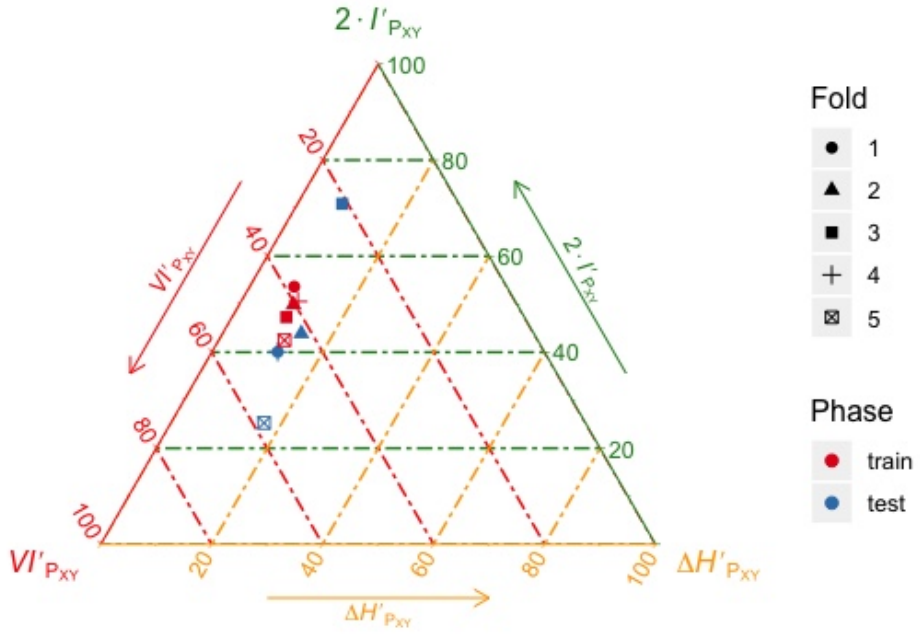


Fig. 5.13. Entropy Triangle in Knn in Ionosphere using the Autoencoder

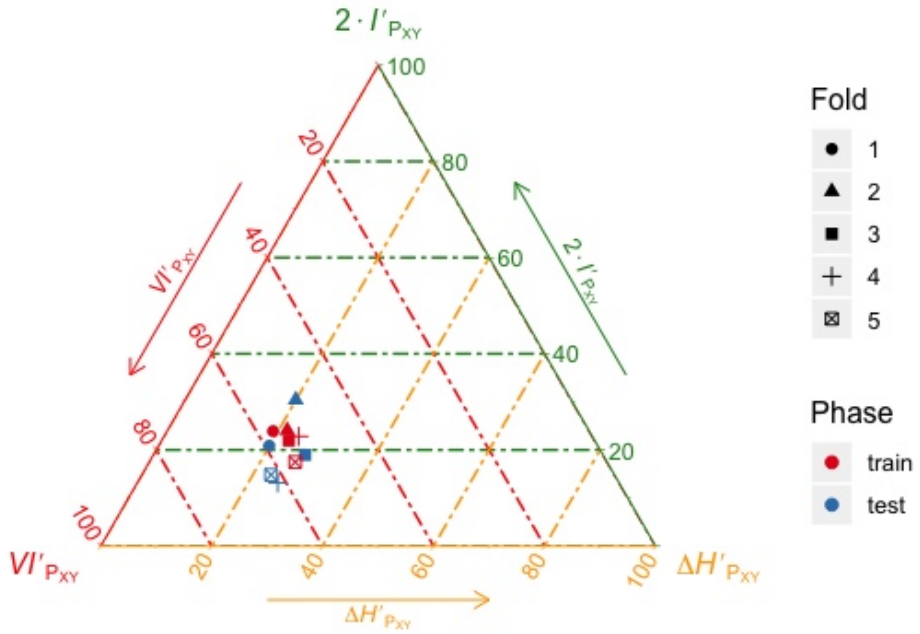


Fig. 5.14. Entropy Triangle in Knn in Ionosphere using the PCA

The results from both Figure 5.13 and Figure 5.14 demonstrate that the heavy unbalanced noticed has affected the outcome of our experiments.

For example, in Figure 5.13 each one of the testing folds has significant differences

in their relative position inside the plot, which translates into scattered points. High variance in our Figures means that our Knn has struggled to do an accurate prediction of our classes.

On the other hand, Figure 5.14 shows that PCA has performed a reliable job at predicting it's classes since all of the points are grouped together in the same area of the ET.

5.2.4. MLP on PCA and Autoencoder

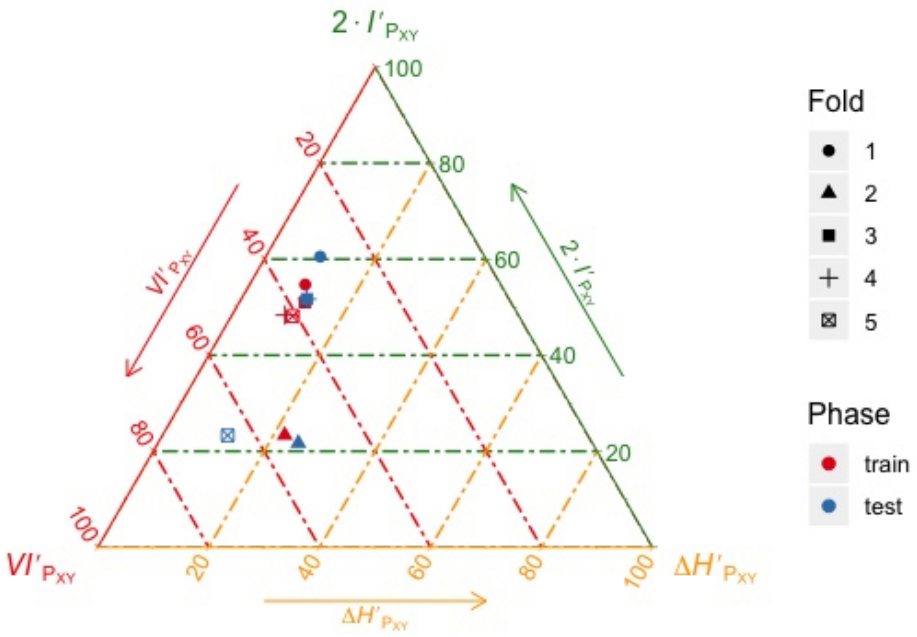


Fig. 5.15. Entropy Triangle in the MLP in Ionosphere using the Autoencoder

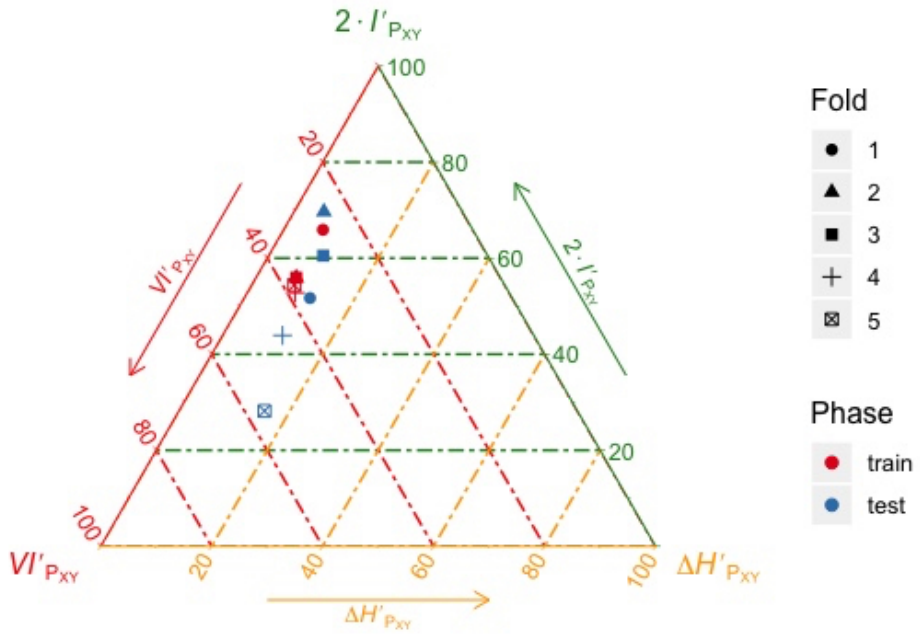


Fig. 5.16. Entropy Triangle in the MLP in Ionosphere using the PCA

By comparing the Folds in Figure 5.15 and Figure 5.16, it is noticeable that the MLP classification process has struggled to find the appropiate predictor for the Folds on both the PCA and the Autoencoder. None of them seems to produce

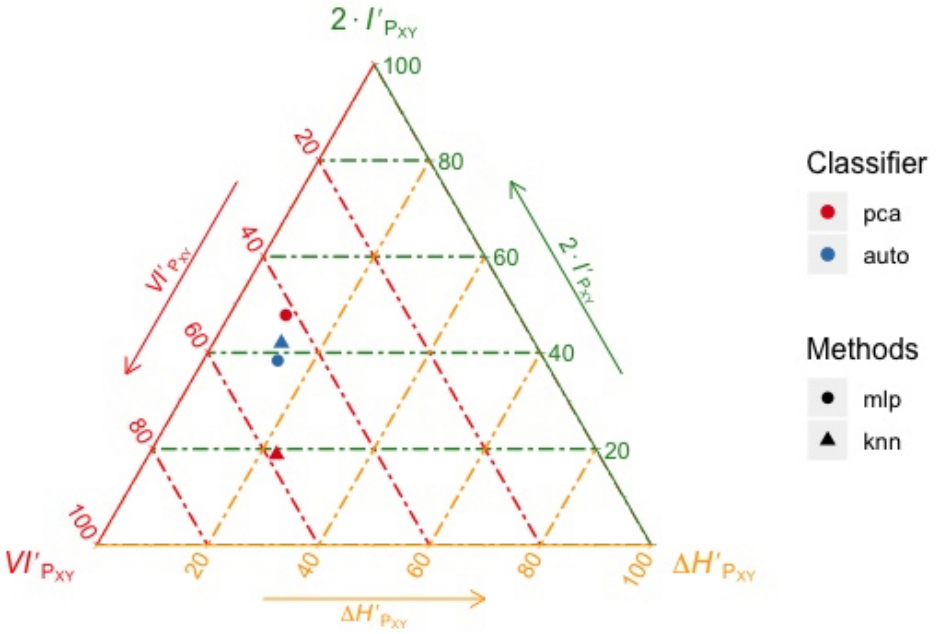


Fig. 5.17. Entropy Triangle in Ionosphere with the total test fold value

5.3. MNIST Dataset

5.3.1. Data Preparation

5.3.2. 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 considered for the sake of this project.

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

Table 5.4. 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 5.5. 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 succesful to the transference of information.

5.3.3. 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 5.4 and Table 5.5, 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 5.6. 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 5.7. MNIST MLP LAYERS OF THE PCA

As we can see on Table 5.7 the PCA MLP has the advantage of being compossed by a greater amount of neurons, making it more computationally powerful and thus having an advantage towards the smaller Autoencoder MLP.

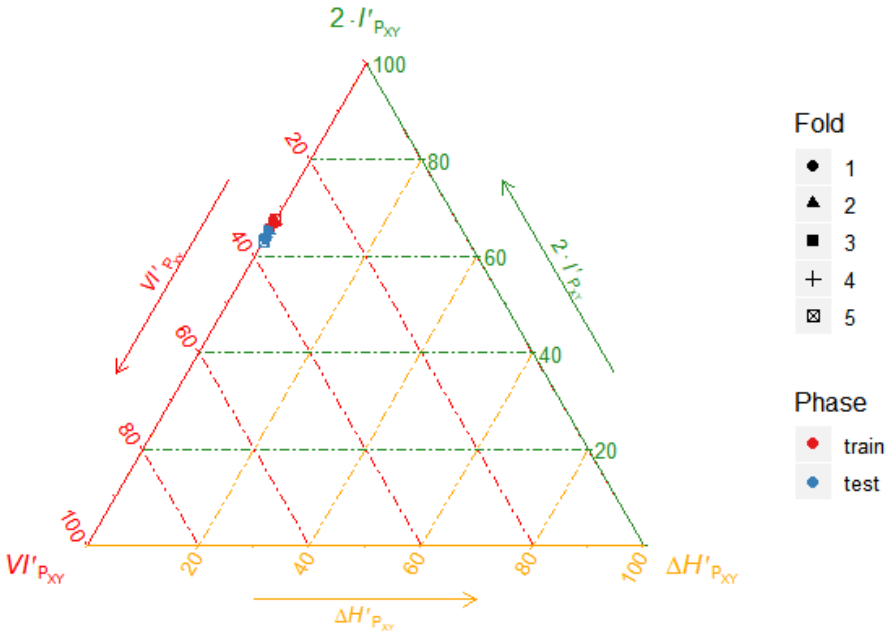


Fig. 5.18. Entropy Triangle in MLP in MNIST using the Autoencoder

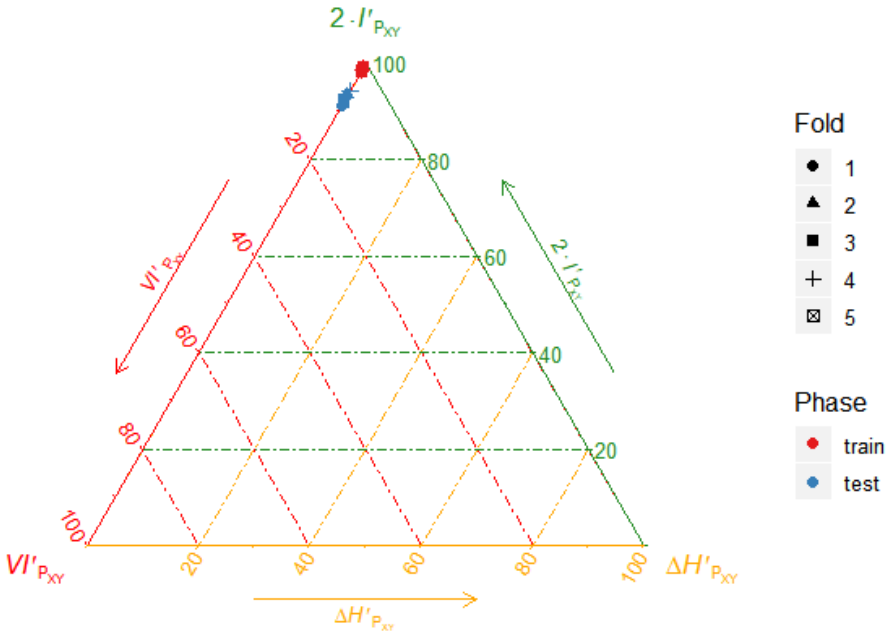


Fig. 5.19. Entropy Triangle in MLP in MNIST using the PCA

Once we results have been obtained, Figure 5.19 shows an almost perfect classification being performed at the MLP. Figure 5.19 proves that the Autoencoder has been outperformed by the MLP, although it is still doing a good job at classifying the numbers.

Both of them also are in the straight line ——- , which means that mutual Information transmission is high.

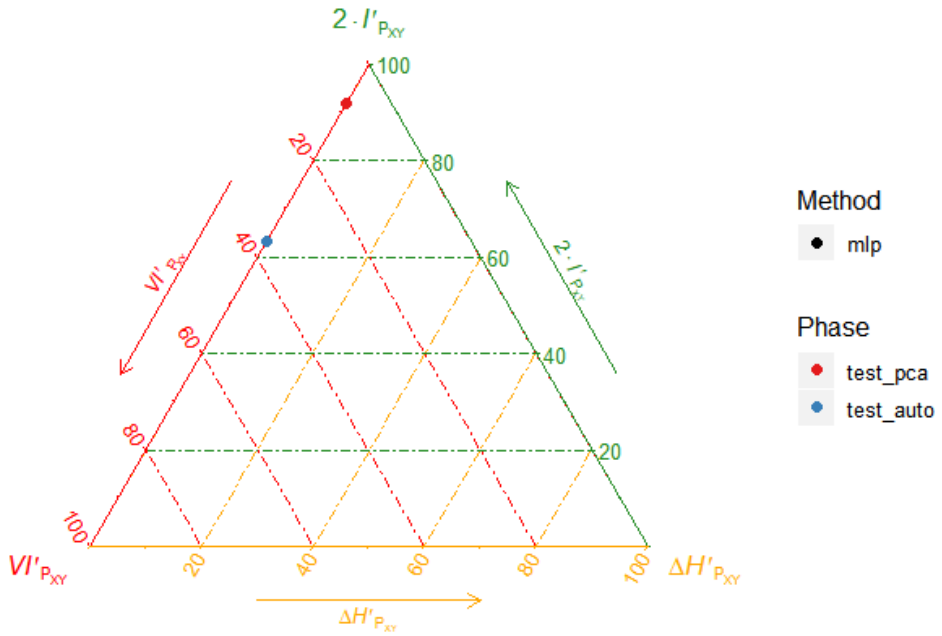


Fig. 5.20. Entropy Triangle in MLP in MNIST total testing result

If we also take a look at Figure 5.19 we get the confirmation about the assesments made about the classification: The point corrresponding to the testing entropy information for the MNIST PCA has placed higher and thus closer to the perfect classifier in the triangle.

6. CONCLUSION