INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Gabriel Barbosa Martinz

USE OF GENERATIVE NEURAL NETWORKS FOR INSTANCE SPACE CODIFICATION AND GENERATION OF DATA WITH SPECIFIC PROPERTIES

Final Paper
2023

Course of Computer Engineering

CDU 681.3:16

Gabriel Barbosa Martinz

USE OF GENERATIVE NEURAL NETWORKS FOR INSTANCE SPACE CODIFICATION AND GENERATION OF DATA WITH SPECIFIC PROPERTIES

Advisor

Prof^{a}. Dr^{a}. Ana Carolina Lorena (ITA)

COMPUTER ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2023

Cataloging-in Publication Data
Documentation and Information Division

Barbosa Martinz, Gabriel
Use of generative neural networks for instance space codification and generation of data with specific properties / Gabriel Barbosa Martinz.
São José dos Campos, 2023.
47f.

Final paper (Undergraduation study) – Course of Computer Engineering– Instituto Tecnológico de Aeronáutica, 2023. Advisor: Prof^{a}. Dr^{a}. Ana Carolina Lorena.

1. Neural networks. 2. Instance space. 3. GAN. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

BARBOSA MARTINZ, Gabriel. **Use of generative neural networks for instance space codification and generation of data with specific properties**. 2023. 47f. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: Gabriel Barbosa Martinz
PUBLICATION TITLE: Use of generative neural networks for instance space codification and generation of data with specific properties.
PUBLICATION KIND/YEAR: Final paper (Undergraduation study) / 2023

It is granted to Instituto Tecnológico de Aeronáutica permission to reproduce copies of this final paper and to only loan or to sell copies for academic and scientific purposes. The author reserves other publication rights and no part of this final paper can be reproduced without the authorization of the author.

Gabriel Barbosa Martinz
Rua H8B, 203
12.228-461 – São José dos Campos–SP

USE OF GENERATIVE NEURAL NETWORKS FOR INSTANCE SPACE CODIFICATION AND GENERATION OF DATA WITH SPECIFIC PROPERTIES

This publication was accepted like Final Work of Undergraduation Study

Gabriel Barbosa Martinz
Author

Ana Carolina Lorena (ITA)
Advisor

Prof. Dr. Marcos Máximo
Course Coordinator of Computer Engineering

São José dos Campos: June 23, 2023.

Acknowledgments

To my parents, Michelle and Raimundo, for always supporting me in my academic journey. To my advisor, Prof. Ana Carolina Lorena for the guidance provided over this research. To my beloved Vitória, for the love and emotional support she gives to me. To my roommates over these years for being great friends. To the friends I've made when at ITA and before it for helping me and making my undergraduate journey more fun. To all the amazing professors I had over my undergraduate studies for shaping my interests in engineering.

"Eu quase que nada não sei. Mas desconfio de muita coisa."
— Guimarães Rosa

Abstract

One topic of study in Machine Learning is the study of algorithmic performance and which methodologies may be used to assess this performance. A methodology known as Instance Space Analysis has been used to relate predictive performance in classification algorithms to instance hardness (how hard an instance is for an algorithm to classify). The original methodology has been defined with the instance being an entire dataset, but further work has been made to make the instance as fine-grained as an individual observation. In this work we will build upon this methodology and we propose the creation of a generative neural network model to generate new observations for a classification algorithm with predefined hardness properties.

List of Figures

List of Tables

Contents

1 Introduction

1.1 Motivation

Often in a problem being tackled with Machine Learning (ML) techniques one of the most important part of the solving process is the algorithm selection. Each algorithm has a specific bias which makes it more suitable for some classes of problems than others (Adam *et al.*, 2019).

It is desirable, then, that we may have a way of measuring the relationship of the performance of a given algorithm in a problem with the problem's characteristics, since knowing which data is easy or difficult for a given model to classify is useful in the way that we may make changes to the original model.

Muñoz *et al.* (2018) has introduced a methodology called Instance Space Analysis (ISA), a novel way of performance evaluation and algorithm selection in classification problems by mapping the statistical properties of an instance (an entire dataset) into how difficult the instance is for the classification algorithm to perform. Further, in Paiva *et al.* (2022), the methodology has been modified to have a more fine-grained analysis, with the instance being reduced to an individual observation in a classification dataset.

Given this, we can map each observation into a hardness level. But one feature of the ISA not investigated in the instance-level proposal of Paiva *et al.* (2022) is how to generate new problem instances with desired characteristics in order to expand the dataset with instances that may defy the classification techniques. This project aims to fill this gap.

One type of model that may give us this feature from this data is a neural network that encodes the data into the instance space and decodes from it. Using this approach, we can then generate data in a region of the instance space and use the decoding part of the network to infer a reconstruction of the original data and find difficulty metrics from it, as well as be able to challenge the classification techniques at different levels. We can use this to verify how the original model behaves with data with a given instance space profile or to challenge the model, such as generating adversarial examples (Yuan *et al.*, 2019).

1.2 Objective

This work's objective lies in providing a framework for data generation based on the instance space generated for the data and monitor the original model's behaviour using the generated data.

1.3 Scope

The scope of this work will be limited to exploring an encoder-decoder implementation for the generation of data. The modelling will be made entirely using Python, with the PyTorch (Paszke *et al.*, 2019) framework. PyHard (Paiva *et al.*, 2022) will be used for reproducing the ISA methodology.

1.4 Outline of this work

The document is organized as follows:

- Chapter 2 introduces some Machine Learning terminology and describes the encoder and decoder neural network models;
- Chapter 3 describes the ISA framework;
- Chapter 4 presents the methodology to be followed in the computational experiments;
- Chapter 6 concludes the work presenting future activities and a work schedule.

2 Machine Learning

This chapter will introduce Machine Learning (ML) concepts and techniques being explored in this work, namely the classification problem, feature learning, neural networks and encoder and decoder neural networks.

2.1 Classification

In Statistics and Machine Learning, a problem is defined as a classification problem when it consists in identifying to which categories a member of a population belongs to. An example might be identifying which race of domestic cat is shown in a picture containing a cat (Kolla, 2020). An algorithm that implements classification is known as a classifier. The classifier works by analysing each observation into dependent variables and either mapping those to the categories or by comparing each observation to previous observations by means of a similarity function or loss function (Faceli *et al.*, 2021).

Terminology between Statistics and Machine Learning tend to differ. In this work, we will be using the terminology found in Machine Learning, namely:

- dependent variables are called features;
- categories are called classes;

In this work, we will not focus on a specific classification algorithm since ISA is not dependent on the algorithm used, only on the problem of classification.

2.2 Feature learning

Feature learning, or representation learning(Bengio *et al.*, 2013), is a set of techniques that allow a model to discover representations needed for feature detection or classification from raw data. This is a method by which we replace feature engineering and have the model both learn the features and use them to perform the task.

Feature learning can be either supervised, unsupervised or self-supervised:

- in supervised learning, features are learned using labeled input data;
- in unsupervised learning, features are learned using the relationship between the individual observations in the input data. Examples include principal component analysis and various forms of clustering;
- in self-supervised learning, features are learned using unsupervised methods and then input-label pairs are constructed, enabling supervised learning. Examples include autoencoders and word embeddings.

2.3 Neural networks

Neural networks, formally called *artificial neural networks* (ANNs), are computational models inspired by networks of biological neurons (Puri *et al.*, 2016). They are made up of multiple nodes called artificial neurons that map an input to an output based on mathematical operations. This model is used extensively in ML applications because of its perceived intelligent behaviour that comes from the interactions between neurons.

2.3.1 Artificial neuron

The artificial neuron is the most basic block of an ANN. It maps inputs to an output in the given fashion:

$$y = f(\mathbf{w} \cdot \mathbf{x} + b), \tag{2.1}$$

where the symbols are defined as:

$$\begin{array}{ll} \mathbf{x} = [x_1, x_2, \dots, x_n] & \text{Input vector;} \\ \mathbf{w} = [w_1, w_2, \dots, w_n] & \text{Weight vector;} \\ f & \text{Activation function;} \\ y & \text{Neuron output;} \\ b & \text{Bias,} \end{array} \tag{2.2}$$

Figure 2.1 shows the artificial neuron model. This model of neuron is useful because it incorporates both the linear combination of input values and bias and the non-linearity of the activation function, which means it may function as a part of an universal function approximator (Hornik *et al.*, 1989).

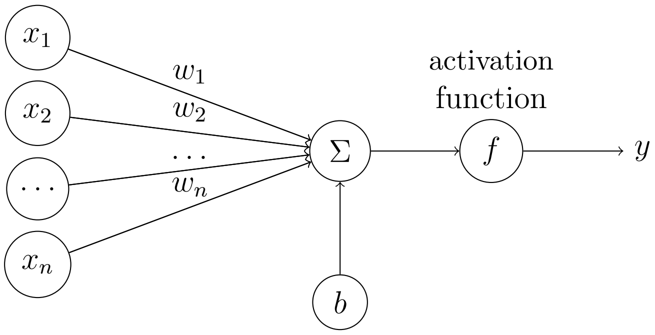

FIGURE 2.1 – Model of an artificial neuron.

2.3.2 The network

As said before, an ANN is a network of artificial neurons. Such network may be built by having the neurons configured in layers, having each neuron in a layer connected only to neurons in either preceding or following layers or with other arrangement of connections. Figure 2.2 shows a simple model of a fully-connected (a neuron in a layer connects to every neuron in the next layer) neural network, having 3 inputs, 4 middle nodes (called a hidden layer) and 3 output nodes.

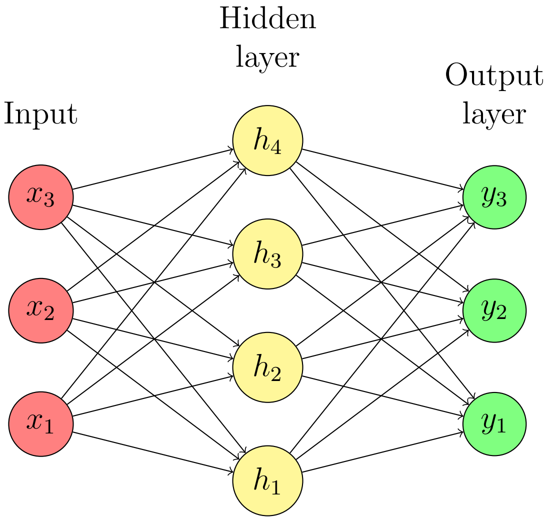

FIGURE 2.2 – A model of a simple fully-connected neural network with 3 inputs, 3 outputs and a hidden layer with 4 nodes.

2.3.3 Learning

Learning is the process by which an ANN adapts itself to a given task using data. It involves adjusting the weights of the network to improve some predefined metric (e.g. accuracy) and minimizing observed errors. In practice, learning is done by defining a loss function which is evaluated during learning and as long as its output (called loss, for short) decreases, the learning continues.

Most learning models are applications of optimization theory, like the gradient descent algorithm. In this algorithm the purpose is to find a local minima by moving against the gradient of the function. It is the basis for the Adam optimizer (Kingma; Ba, 2017), used extensively in ANNs.

2.3.3.1 Learning rate

The learning rate is a parameter in an optimization algorithm, defining the size of each step towards the local minima of a loss function. Higher learning rates shorten the learning time, but at a cost of possibly never converging and higher errors, while setting it at a value too low might have it converging in an undesirable local minimum.

2.3.3.2 Backpropagation

Backpropagation is a method to adjust the weights of the network and minimize the mean squared error. It computes the gradient of the loss function with respects to the weights and propagates backwards from the output layer to the initial layers in order to avoid redundant calculations.

2.4 Encoder, decoder and autoencoder neural networks

Neural networks can be used in feature learning tasks, as shown in section 2.2. We can construct neural networks to learn efficient codings of the input data by means of an encoder network, which is a network that transforms input data into a representation in a latent space. Figure 2.3 shows a representation of an encoder network. A decoder network is much the same as an encoder, but in the opposite direction, transforming the latent space representation into the original data.

An **autoencoder** is one such networks, composed of two parts: an encoder (or *encoding function* f) and a decoder (or *decoding function* g) that recreates the data from this representation. The learning problem is then defined as minimizing a loss function $L(\mathbf{x}, g(f(\mathbf{x})))$ that penalizes $g(f(\mathbf{x}))$ from being too different from $\mathbf{x}$ (Goodfellow *et al.*, 2016). A representation of an autoencoder is shown in figure 2.4.

It is not useful to have the autoencoder copy each input perfectly to the output. Usually they are restricted in ways that allow them to copy only approximately, and to copy only input that resembles the training data (Goodfellow *et al.*, 2016).

One way to restrict the autoencoder is to have the encoder be an undercomplete network, meaning that the dimensionality of the latent space is smaller than the dimen-

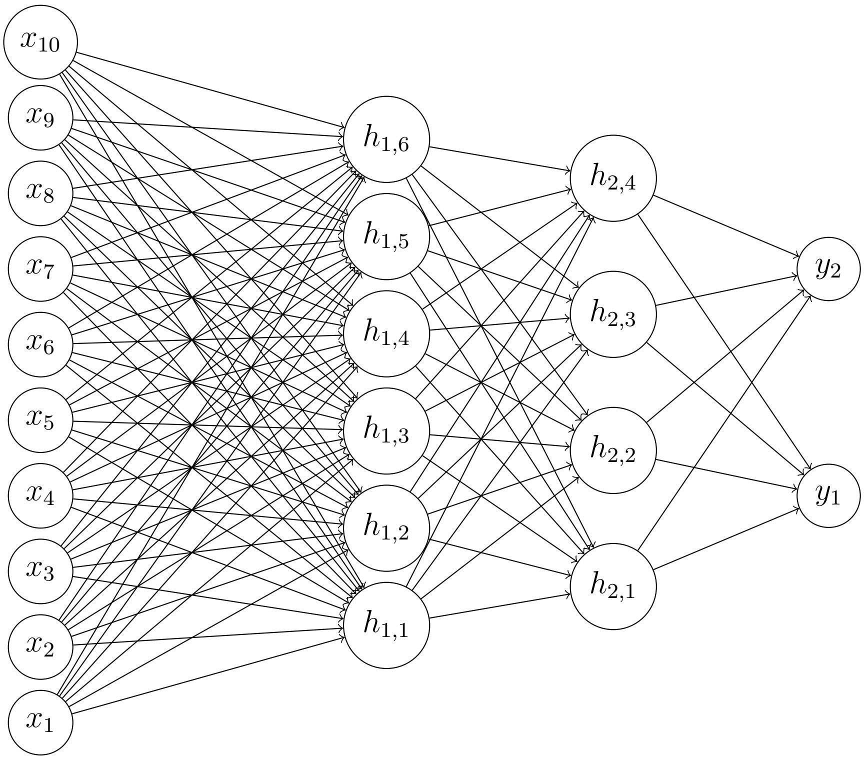

FIGURE 2.3 – Representation of an encoder

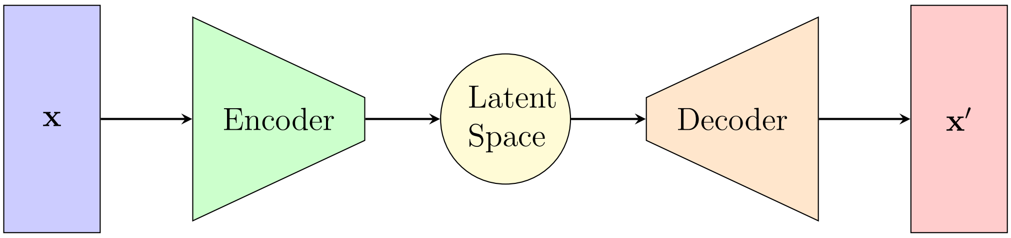

FIGURE 2.4 – Representation of an autoencoder with input $\mathbf{x}$.

sionality of the input space. This forces the autoencoder to learn the most important features of the input data in order to be able to recreate it. In this case, if the decoder is linear and the loss function L is the mean squared error, then the optimal solution to the autoencoder is the principal component analysis (Goodfellow *et al.*, 2016).

Unfortunately, if the encoder and the decoder are allowed too much capacity, the autoencoder will learn to copy the input to the output without learning any useful information about the data distribution (Goodfellow *et al.*, 2016). A similar problem occurs if the latent space is allowed to have the same or greater dimension than the input data.

3 Instance Space Analysis

In this chapter we will be introducing the novel methodology for algorithm selection and performance evaluation called Instance Space Analysis (ISA), introduced in Muñoz *et al.* (2018). We will be showing the original definition of an instance space and the adaptation of the methodology brought up by Paiva *et al.* (2022) relating instance hardness.

3.1 Instance spaces

ISA is, at its core, an extension of the Algorithm Selection Problem (ASP) (Rice, 1976). Figure 3.1 shows the ISA framework with the ASP highlighted in blue. The objective in the ASP is to automate the process of selecting algorithms based on past similar solved problems. The following sets compose the core of the ASP:

- **Problem Space** $\mathcal{P}$: contains all instances of the problem being analysed;
- **Instance Sub-space** $\mathcal{I}$: contains a subset of instances from $\mathcal{P}$ for which the characteristics and solutions are available;
- **Feature Space** $\mathcal{F}$: a set of descriptive characteristics of the instances in $\mathcal{I}$. These are also known as meta-features;
- **Algorithm Space** $\mathcal{A}$: contains algorithms that may be used to solve the instances in $\mathcal{I}$;
- **Performance Space** $\mathcal{Y}$: contains the performance evaluations of the algorithms in $\mathcal{A}$ over the instances in $\mathcal{I}$.

The combination of tuples $(x, f(x), \alpha, y(\alpha, x))$, where $x \in \mathcal{I}$, $f(x) \in \mathcal{F}$, $\alpha \in \mathcal{A}$ and $y(\alpha, x) \in \mathcal{Y}$, composes a meta-dataset $\mathcal{M}$. A meta-learner S can then be trained to select the best algorithm for an instance x based on its meta-features, that is, an algorithm α^* with maximum predictive performance for x as given by y:

$$\alpha^* = S(f(x)) = \arg\max_{\alpha} ||y(\alpha, x)||. \tag{3.1}$$

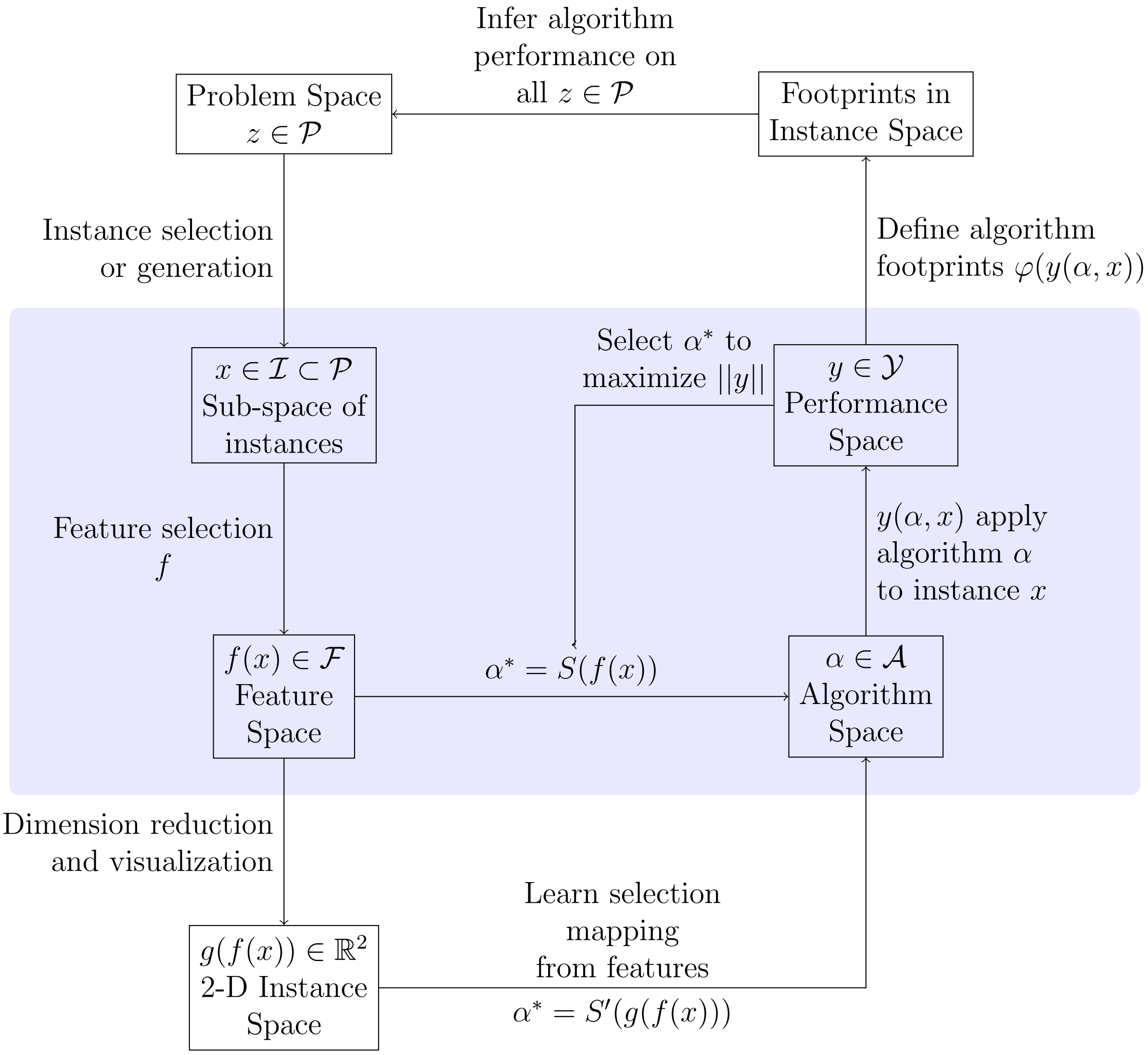
Infer algorithm performance on all $z \in \mathcal{P}$
Problem Space $z \in \mathcal{P}$
Footprints in Instance Space
Instance selection or generation
Define algorithm footprints $\varphi(y(\alpha, x))$
$x \in \mathcal{I} \subset \mathcal{P}$ Sub-space of instances
Select α^* to maximize $||y||$
$y \in \mathcal{Y}$ Performance Space
Feature selection f
$y(\alpha, x)$ apply algorithm α to instance x
$f(x) \in \mathcal{F}$ Feature Space
$\alpha^* = S(f(x))$
$\alpha \in \mathcal{A}$ Algorithm Space
Dimension reduction and visualization
Learn selection mapping from features
$g(f(x)) \in \mathbb{R}^2$ 2-D Instance Space
$\alpha^* = S'(g(f(x)))$

FIGURE 3.1 – ISA framework. Extracted from Muñoz *et al.* (2018). The ASP is highlighted in blue.

The ISA framework goes further to give insights into why some instances are harder to solve than others, using both the information of meta-features and algorithmic performance in a 2-D embedding, called Instance Space (IS), that can be visually inspected. An optimization problem is solved to find the mapping from meta-features to the IS $g(f(x))$, such that the distribution of algorithmic performance metrics and meta-features values display as close a linear trend as possible in the IS embedding. This embedding can then be inspected for regions of good and bad algorithmic performance and a new learner can be created to select new algorithms, as in:

$$\alpha^* = S'(g(f(x))) \tag{3.2}$$

In the IS, it is also possible to define algorithm footprints $\varphi(y(\alpha, x))$, which are areas where an algorithm α is expected to perform well. A set of objective measures can be derived from these footprints and aid in the inference of algorithmic performance for other instances that were not in $\mathcal{I}$, such as:

- the area of the footprint A, which can be normalized across multiple algorithms for comparison;
- the density of the footprint ρ, which can be calculated as the ratio between the number of instances enclosed by the footprint and its area;
- the purity of the footprint p, which is the percentage of instances in the footprint that have good performance.

Summarizing, the application of ISA requires (Muñoz *et al.*, 2018):

1. building the meta-dataset $\mathcal{M}$;
2. reducing the set of meta-features in $\mathcal{M}$, keeping only those able to discriminate algorithmic performance;
3. creating the 2-D IS from $\mathcal{M}$;
4. building the algorithms' footprints in the IS.

3.1.1 Instance Space construction

The problem of finding the optimal mapping between the instance metadata domain to a 2-D IS has been modelled using the Prediction Based Linear Dimensionality Reduction (PBLDR) method (Smith-Miles; Muñoz, 2023). Given a meta-dataset $\mathcal{M}$ with m meta-features, n instances and a algorithms, let $F \in \mathbb{R}^{m \times n}$ be the matrix containing the meta-features values for all instances and $Y \in \mathbb{R}^{n\ timesa}$ be the matrix containing the performance measure of the algorithms on the same instances. The optimal projection is achieved by minimizing the MSE:

$$MSE = ||\mathbf{F} - \hat{\mathbf{F}}||_F^2 + ||\mathbf{Y} - \hat{\mathbf{Y}}||_F^2, \tag{3.3}$$

such that:

$$\mathbf{Z} = \mathbf{A}_r\mathbf{F}, \tag{3.4}$$

$$\hat{\mathbf{F}} = \mathbf{B}_r\mathbf{Z}, \tag{3.5}$$

$$\hat{\mathbf{Y}}^T = \mathbf{C}_r\mathbf{Z}. \tag{3.6}$$

Where $\mathbf{A}_r \in \mathbb{R}^{2\times m}$, $\mathbf{B}_r \in \mathbb{R}^{m\times 2}$ and $\mathbf{C}_r \in \mathbb{R}^{a\times 2}$. $\mathbf{Z} \in \mathbb{R}^{2\times n}$ is the matrix of instance coordinates in the IS and A_r is the projection matrix. Smith-Miles & Muñoz (2023) solves this problem numerically using the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm.

3.1.2 Footprint analysis

As said before, a footprint is a region in the IS where an algorithm is expected to perform well based on inference from empirical performance analysis (Muñoz *et al.*, 2018). Smith-Miles & Muñoz (2023) goes into more detail of its construction, qualitatively we can infer from the set of footprint measures defined before that a good algorithm will have large quantities of each of those measures, i.e. a large area shows that an algorithm performs well in a large portion of the IS, a large density shows that the footprint contains a large amount of instances and a large purity shows that the algorithm performs well in most instances.

3.2 ISA for a single dataset

In Muñoz *et al.* (2018), ISA is defined with an instance being an entire classification dataset. Paiva *et al.* (2022) has adapted the framework and reduced the problem space $\mathcal{P}$ into a single dataset, with each instance being an observation inside the dataset. This has come with the removal of some steps of the original ISA, namely the algorithmic recommendation module and the creation of new instances from the IS.

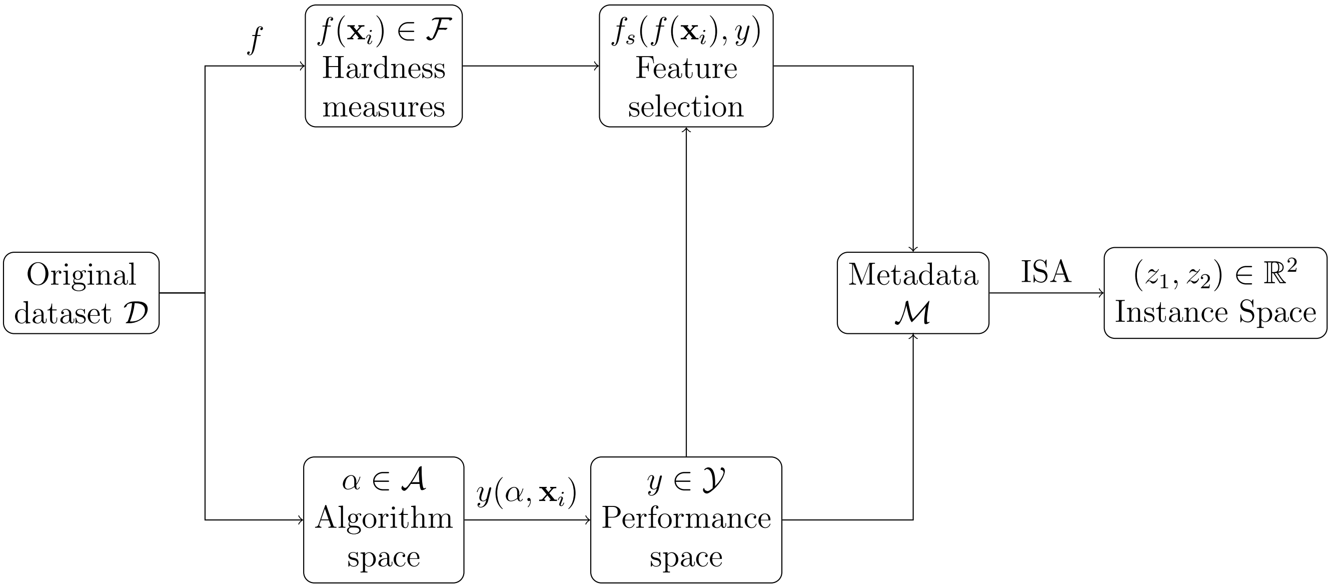

FIGURE 3.2 – ISA framework for a single dataset. Extracted from Paiva *et al.* (2022).

Given a classification dataset $\mathcal{D}$ with n_D instances $\mathbf{x}_i$, m_D input features and each instance labelled in a class y_i, we have:

- **Problem space** $\mathcal{P}$: reduced to the dataset $\mathcal{D}$;
- **Instance sub-space** $\mathcal{I}$: contains all individual instances $\mathbf{x}_i$;
- **Feature space** $\mathcal{F}$: contains a set of meta-features known as *hardness measures.*

The modified framework is shown in Figure 3.2. For each instance, a set of hardness measures are stored in $\mathcal{F}$, and each algorithm $\alpha \in \mathcal{A}$ is evaluated over the instance $\mathbf{x}_i$ and has its performance $y(\alpha, \mathbf{x}_i)$ measured and stored in $\mathcal{Y}$. The measure used is a cross-validated log-loss error obtained in the prediction of the instance label. A feature selection is made with these measures and the hardness measures, resulting in a reduced feature set $f_s(f(\mathbf{x}_i), y)$.

Combining the reduced feature set to the predictive performance of the algorithms allows us to construct the meta-dataset $\mathcal{M}$, from which the IS is extracted. The steps are explained in (Paiva *et al.*, 2022), we will summarize some important topics.

3.2.1 Hardness measures

(Paiva *et al.*, 2022) revisits the definition of *Instance Hardness* (IH), a property that indicates the probability of an instance being misclassified given a classification hypothesis (Smith *et al.*, 2014):

$$IH_h(\mathbf{x}_i, y_i) = 1 - p(y_i|\mathbf{x}_i, h), \tag{3.7}$$

where $h : X \to Y$ is a classification hypothesis mapping an input space of features X to an output space of labels Y. In practice, h is a learning algorithm induced from the dataset $\mathcal{D}$ and hyperparameters β, that is $h = l(\mathcal{D}, \beta)$. We can then compute the IH of the whole algorithm space $\mathcal{A}$:

$$IH_{\mathcal{A}}(\mathbf{x}_i, y_i) = 1 - \frac{1}{|\mathcal{A}|} \sum_{l \in \mathcal{A}} p(y_i|\mathbf{x}_i, l(\mathcal{D}, \beta)). \tag{3.8}$$

The idea is that instances frequently misclassified over the algorithm space can be considered hard, and instances being correctly classified are considered easy. Paiva *et al.* (2022) also shows a set of *hardness metrics* being extracted for each instance. Table 3.1 lists the measures used, the formal definition for each one is shown in Appendix A.

Measure	Acronym	Min.	Max.
k-Disagreeing Neighbours	kDN	0	1
Disjunct Class Percentage	DCP	0	1
Tree Depth (pruned)	TD_P	0	1
Tree Depth (unpruned)	TD_U	0	1
Class Likelihood	CL	0	1
Class Likelihood Difference	CLD	0	1
Fraction of features in overlapping areas	$F1$	0	1
Fraction of nearby instances different class	$N1$	0	1
Ratio of intra-extra class distances	$N2$	0	≈ 1
Local set cardinality	LSC	0	1
Local set radius	LSR	0	1
Usefulness	U	≈ 0	1
Harmfulness	H	0	≈ 1

TABLE 3.1 – Hardness measures used in Paiva *et al.* (2022).

3.2.2 Performance assessment

To assess the performance from each algorithm in the dataset $\mathcal{D}$, it is first split into r folds according to the cross-validation strategy such that each instance from the dataset belongs to only one fold. $r-1$ folds are used for training the algorithm while the last fold is used for testing. Therefore we can compute a performance estimate for each instance and algorithm combination. The measure used for performance assessment was the log-loss, or cross-entropy, defined in equation 3.9:

$$\text{logloss}(\mathbf{x}_i) = -\sum_{c=1}^{C} y_{i,c} \log(p_{i,c}), \tag{3.9}$$

where C is the number of classes the problem has, $y_{i,c}$ is a binary indicator of whether the class c is an actual label of $\mathbf{x}_i$ or not and $p_{i,c}$ is the probability that the classifier attributes $\mathbf{x}_i$ to class c.

3.2.3 Feature selection

For feature selection, Paiva *et al.* (2022) employs a method called *Minimum Redundancy Maximum Relevance* (MRMR), a criterion that gradually reduces the effect of feature redundancy as more features are selected. There is a trade-off between minimizing the number of redundant features and keeping relevant features, this method rejects redundant features at first but tolerates redundancy as more informative features are added. The mathematical model for MRMR is in equation 3.10:

$$J(\mathbf{f}_k) = MI(\mathbf{f}_k; \mathbf{y}_j) - \frac{1}{|\mathcal{S}_j|} \sum_{\mathbf{f}_i \in \mathcal{S}_j} MI(\mathbf{f}_k; \mathbf{f}_i), \tag{3.10}$$

where $J(\mathbf{f}_k)$ is a score for the k-th feature vector $\mathbf{f}_k$, $MI(\mathbf{f}_k; \mathbf{y}_j)$ is the mutual information between the feature vector and the response variable for the j-th algorithm and $\mathcal{S}_j$ is the set of selected features for the j-th algorithm. $\mathcal{S}_j$ is initially empty and the first feature chosen is the one with maximum mutual information between the feature vector and the response variable $\mathbf{y}_j$. Features are selected based on their score in subsequent rounds until there are a desired number of features chosen $n_f = |\mathcal{S}_j|$.

3.2.4 IS representation and footprints

Given the now constructed meta-dataset $\mathcal{M}$ after feature selection and performance assessments, the 2-D IS can be constructed. Paiva *et al.* (2022) added a rotation to the IS so that bad instances are placed towards the upper left of the IS, while good instances are placed towards the bottom right.

All of this section is implemented in the Python library PyHard (Paiva *et al.*, 2022). This work will use this implementation for reproducing this methodology.

4 Methodology

In this chapter, we will describe the problem's modelling and the main tools being used.

4.1 Environment

The environment used for this work is a computer with an Intel i5-8300H CPU, 16GB of RAM and an NVIDIA GeForce GTX 1050 GPU with 4GB of VRAM. The operating system is Arch Linux.

4.2 Data

In this work, we will be using data for COVID hospitalizations in the city of São José dos Campos. The data is structured as tabular data with most columns encoded as binary values. The columns are:

1. `Fever`: If the patient had fever or not;
2. `Cough`: If the patient had cough or not;
3. `Sore throat`: If the patient had sore throat or not;
4. `Dyspnea`: If the patient had dyspnea or not;
5. `Respiratory.distress`: If the patient had respiratory discomfort or not;
6. `Oxygen.saturation`: If the patient had low oxygen saturation or not;
7. `Diarrhea`: If the patient had diarrhea or not;
8. `Vomit`: If the patient was vomiting or not;
9. `Other.symptoms`: If the patient had other symptoms or not;

10. `Chronic.cardiovascular.disease`: If the patient had chronic cardiovascular disease or not;

11. `Immunodeficiency.immunodepression`: If the patient had immunodeficiency/immunodepressio or not;

12. `Diabetes.mellitus`: If the patient had diabetes mellitus or not;

13. `Obesity`: If the patient was obese or not;

14. `Chronic.respiratory.disease`: If the patient had chronic respiratory disease or not;

15. `Other.risks`: If the patient had other risks not related to the ones before or not;

16. `Sex`: Whether male or female;

17. `Age`: The patient's age;

18. `Hospitalization`: If the patient was hospitalized or not;

4.3 Instance Space generation

For the IS generation, we will use the PyHard (Paiva *et al.*, 2022) library. The library has some examples of the application. We will use a redued version of the configuration, with less hardness measures and algorithms for computational reasons for the hardware defined in Section 4.1. The measures used are:

1. `DCP`: Disjunct class percentage;

2. `F1`: Fraction of features in overlapping areas;

3. `TD_P`: Tree depth using pruned decision trees;

4. `TD_U`: Tree depth using unpruned decision trees;

5. `CL`: Class likelihood;

6. `CLD`: Class likelihood difference;

7. `LSC`: Local set cardinality;

8. `LSR`: Local set radius;

9. `H`: Harmfulness;

10. `U`: Usefulness;

The algorithms used are:

1. Linear Support Vector Classifier;
2. Random forest;
3. Logistic regression;
4. Bagging;

With the original data we generate its IS, resulting in coordinates (z_1, z_2) for each observation. This is the data being fed to the decoder for training.

4.4 Decoder-Encoder creation and loss function

The implementation of the decoder-encoder model is straightforward. Using the PyTorch (Paszke *et al.*, 2019) framework we can define classes that function as neural network models. It has methods for backpropagation and has multiple optimizers implemented. We will be using the Adam optimizer (Kingma; Ba, 2017) for training.

The loss function of the model will need to factor the need of the decoder to generate data that is similar to the original data and the need of the encoder to generate a IS that is similar to the original IS. For this, we will create a loss function for the decoder that needs to factor both continuous and categorical variables and one for the encoder for only the continuous values of $\mathbf{z}$. We will be using for the decoder the sum of the mean squared error (MSE) for the continuous variables and the cross-entropy for the categorical variables. For the encoder, we will be using the MSE. The final loss of the model will be the sum of the losses of the decoder and the encoder.

The training of the model will be done in batches of 250 observations. We will also log the epoch losses in TensorBoard.

4.5 Data generation and re-evaluation of the generated data

For data generation, we will define a region of the IS where we will sample points. Those points will then be normalized under the mean and standard deviation of the original IS and passed into the trained decoder to go from the instance space into the data space. We will then add the generated data into the original data to generate a new IS through PyHard. We will then check if the generated points in the new IS are close to the original generated points.

4.6 Implementation of this methodology

The code that implements this methodology is explained in Appendix B.

5 Results

In this chapter we will show the experiments made and their results, which we will also discuss. Most graphs were generated with Seaborn, the instance space graphics were generated with Bokeh.

5.1 Experiments

In the code, we define an experiment with a YAML configuration file. The file defines the number of output features in each layer, the number of training epochs and parameters for the optimizer. The next sections will define the experiments and show their results.

5.1.1 Original Instance Space and region of data generation

Figure 5.1 shows the original instance space with the parameters defined in Chapter 4. The colorbar in the figure is the instance hardness of each point in the instance space. We will be generating data in the region defined by a rectangle with vertices $(-2.5, -1.0)$ and $(-1.5, 0.0)$.

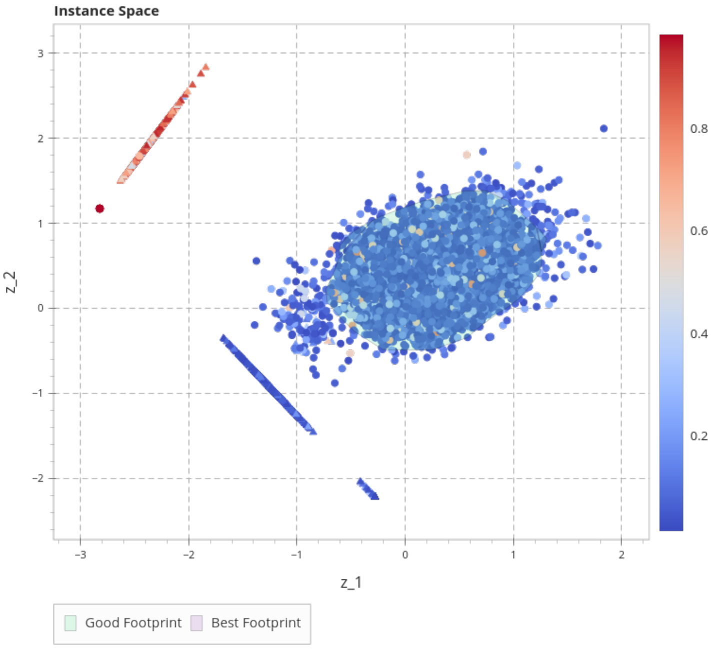

FIGURE 5.1 – Original instance space

5.1.2 Experiment 1

The first experiment is a simple decoder-encoder with 3 layers, with a list of layer output features being $[16, 8, 4]$. The parameters for the optmizer are a learning rate of 0.001 and a weight decay of 0.001, being trained for 50 epochs. The training epoch loss results from TensorBoard are shown in Figure 5.2.

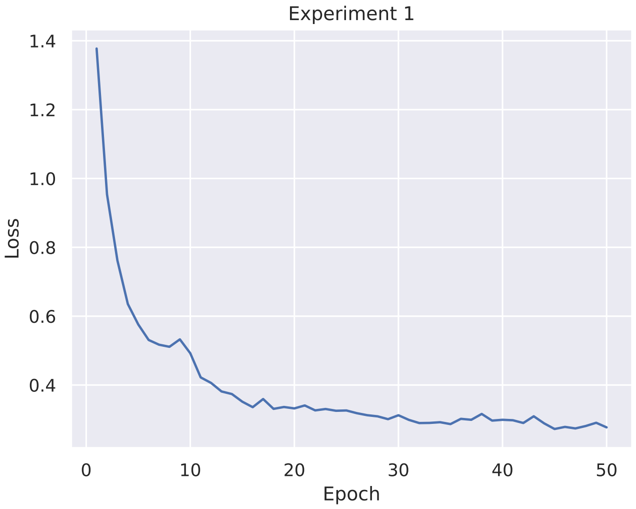

FIGURE 5.2 – Experiment 1 training epoch loss

The loss has reduced nicely over the training epochs. Figure 5.3 shows the instance space with the generated points. The instance space has changed dramatically, but the generated points are not in the position we generated them.

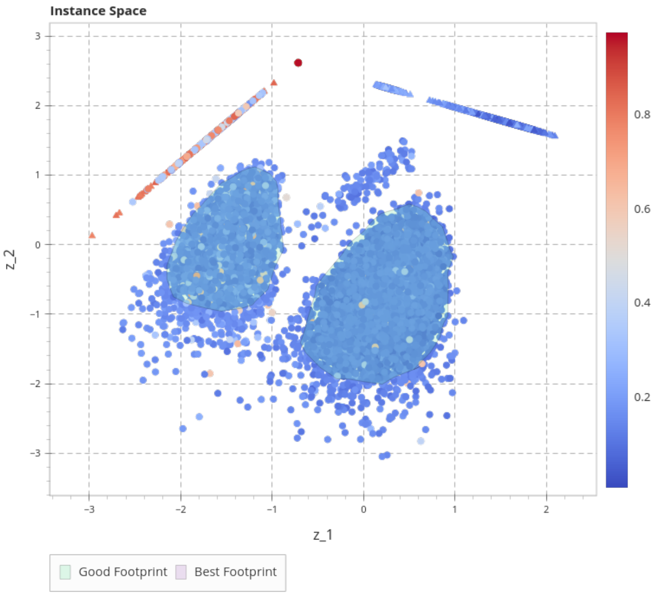

FIGURE 5.3 – Instance space of experiment 1

Figure 5.4 shows an scatterplot of the calculated instance space position value for each dimension versus the original generated value, with the colorbar representing the squared error of each point.

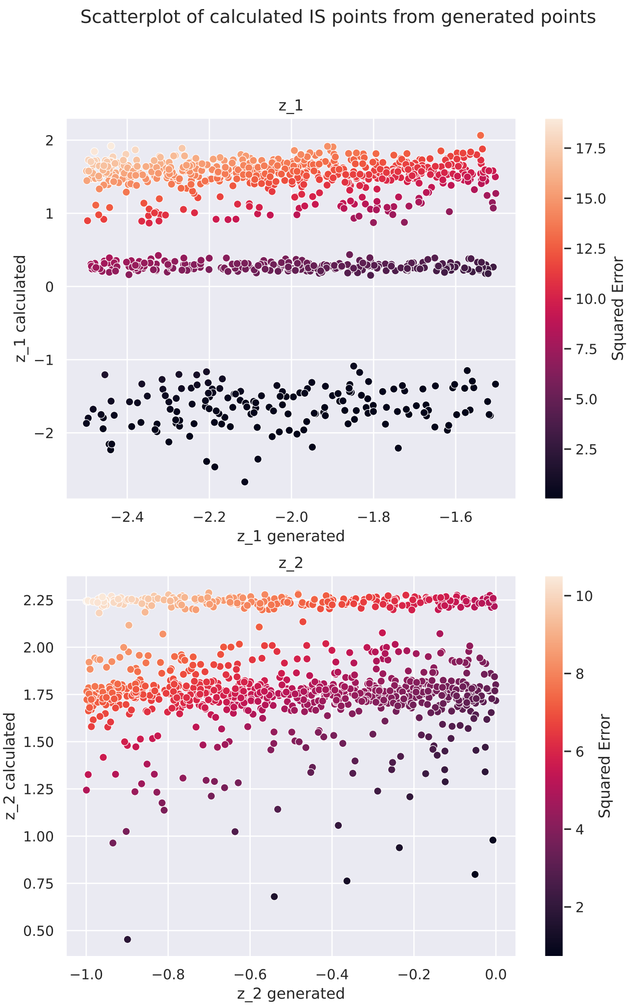

FIGURE 5.4 – Scatterplot of experiment 1

The calculated IS values are bunched in lines, uncorrelated with the original values. This is a sign that the model is not working as intended. Figure 5.5 shows the same scatterplot but for the calculated points against the points returned by the encoder, which are also uncorrelated.

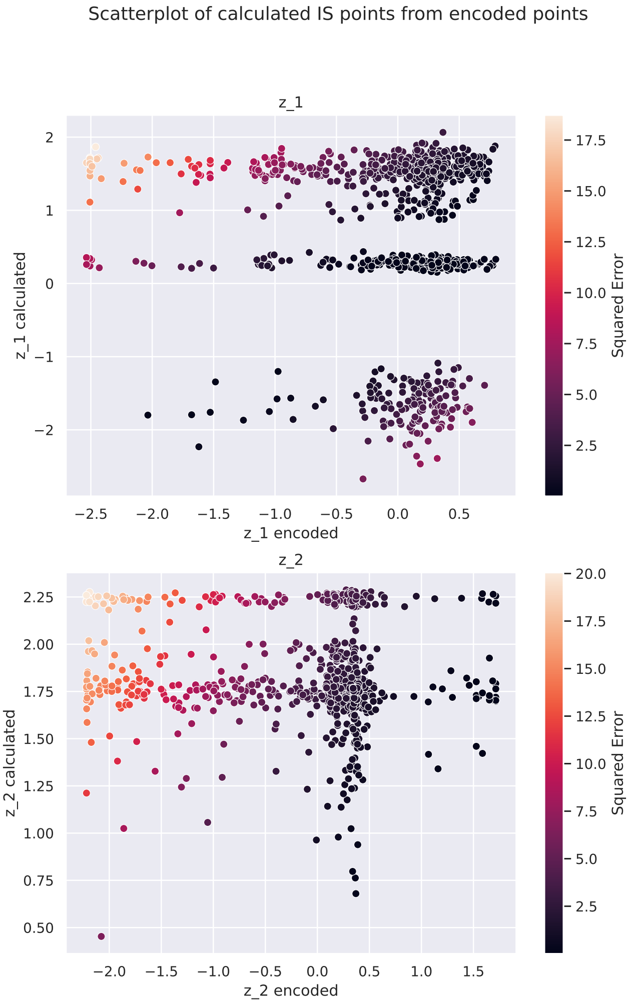

FIGURE 5.5 – Scatterplot of experiment 1 encoder

5.1.3 Experiment 2

In this experiment, we have two more layers and the output features are the list $[128, 32, 16, 8, 4]$. The optimizer parameters are the same, but the training epochs were reduced to 40. The training epoch loss results from TensorBoard are shown in Figure 5.6.

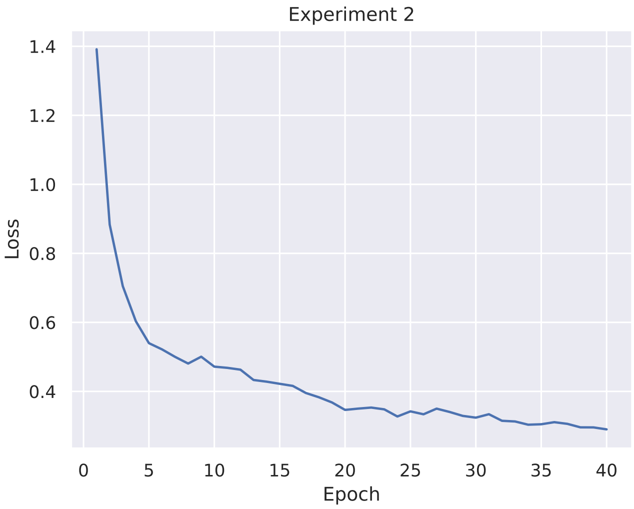

FIGURE 5.6 – Experiment 2 training epoch loss

The loss has reduced nicely over the training epochs as well. Figure 5.7 shows the instance space with the generated points. The instance space has again changed dramatically, with Figure 5.8 showing another scatterplot of the dimensions.

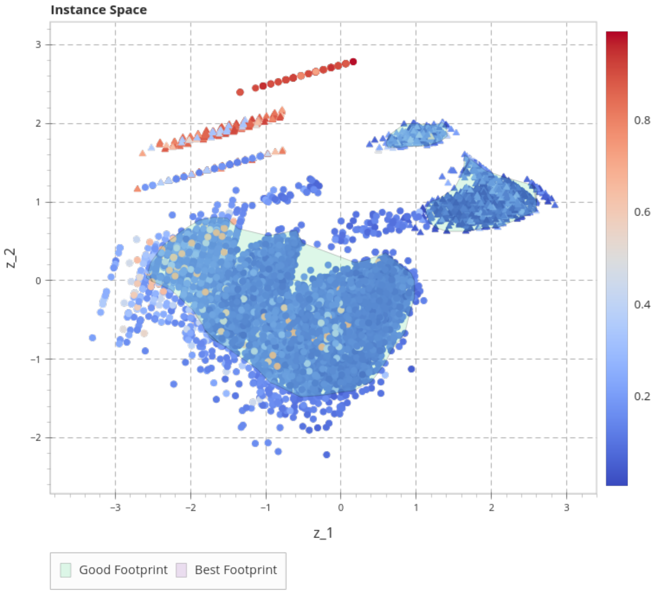

FIGURE 5.7 – Instance space of experiment 2

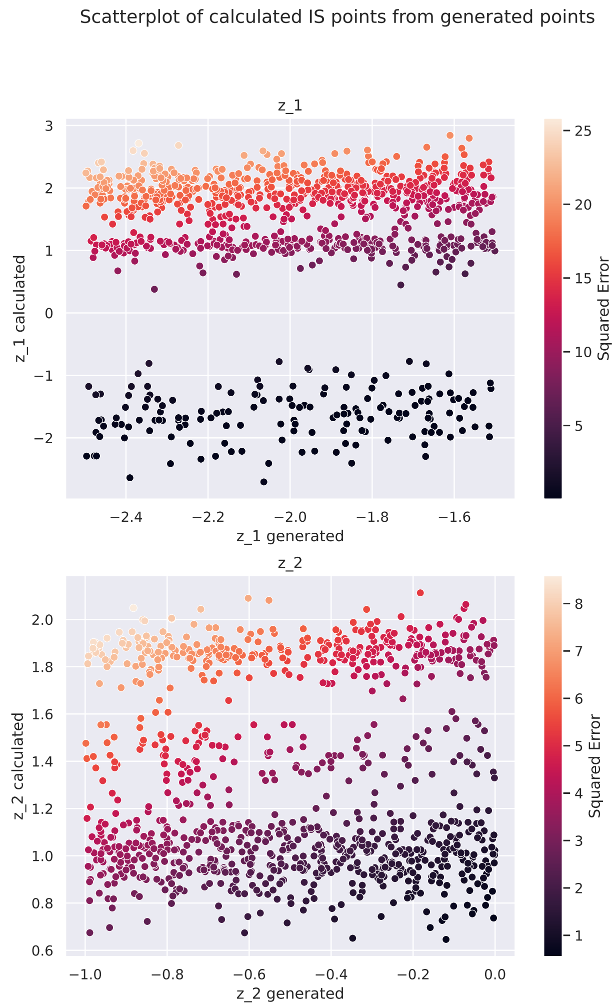

FIGURE 5.8 – Scatterplot of experiment 2

The calculated IS values are again bunched in lines, uncorrelated with the original values. Figure 5.9 shows the scatterplot for the points returned by the encoder. Again, they are uncorrelated.

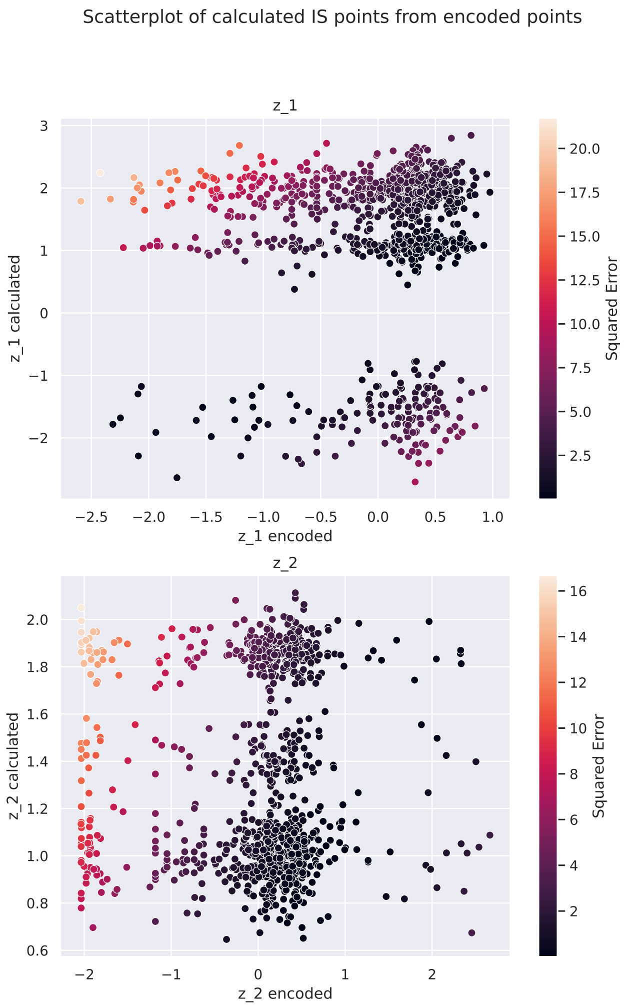

FIGURE 5.9 – Scatterplot of experiment 2 encoder

5.1.4 Experiment 3

In this experiment we have again deepened the network, with 8 layers and list of features $[1024, 1024, 256, 128, 32, 16, 8, 4]$. The optimizer parameters and training epochs are the same as Experiment 2. The training epoch loss results from TensorBoard are shown in Figure 5.10.

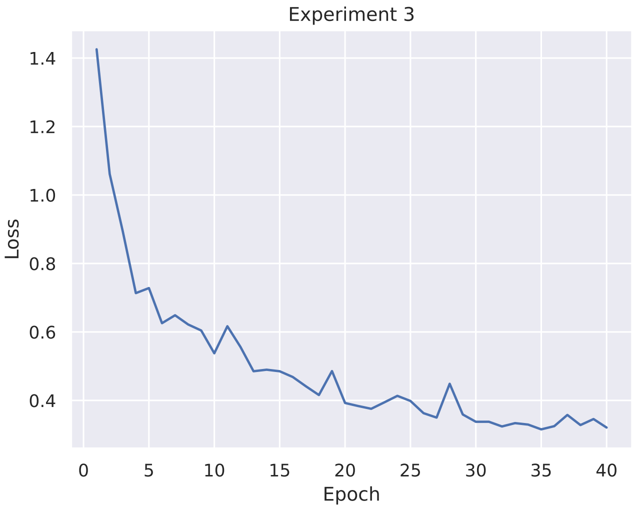

FIGURE 5.10 – Experiment 3 training epoch loss

The loss has again reduced nicely over the training epochs. Figure 5.11 shows the instance space with the generated points. The instance space has again changed dramatically, with Figure 5.12 showing another scatterplot of the dimensions.

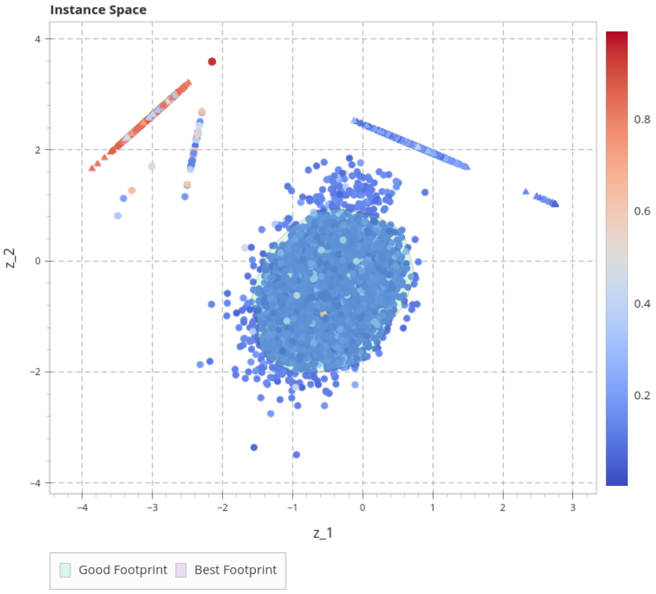

FIGURE 5.11 – Instance space of experiment 3

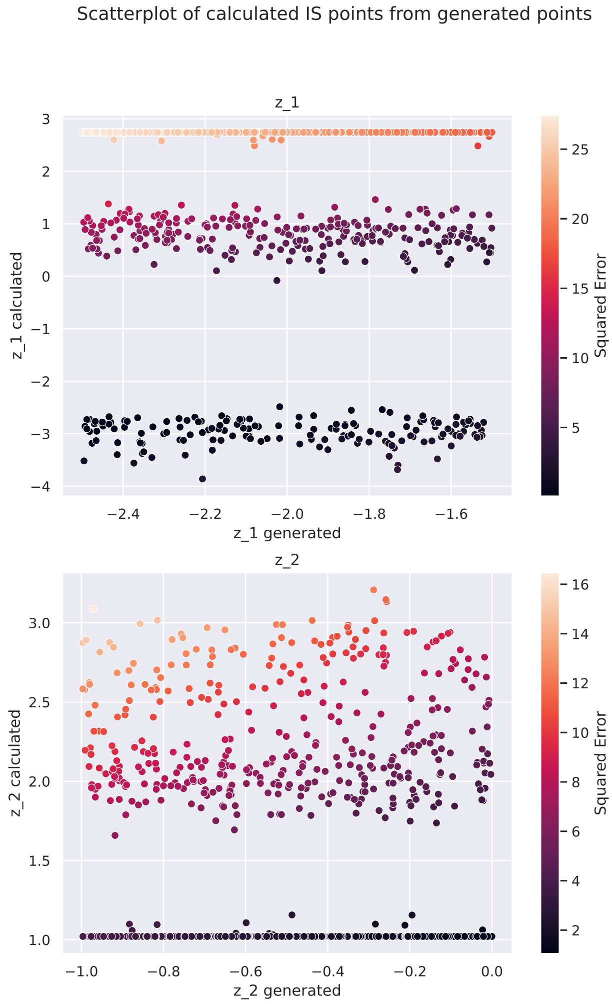

FIGURE 5.12 – Scatterplot of experiment 3

Again, most points are uncorrelated with the original values, bunched up in lines. A curious result has been the points bunched up near $(3, 1)$. Figure 5.13 shows the scatterplot for the points returned by the encoder, which are also uncorrelated.

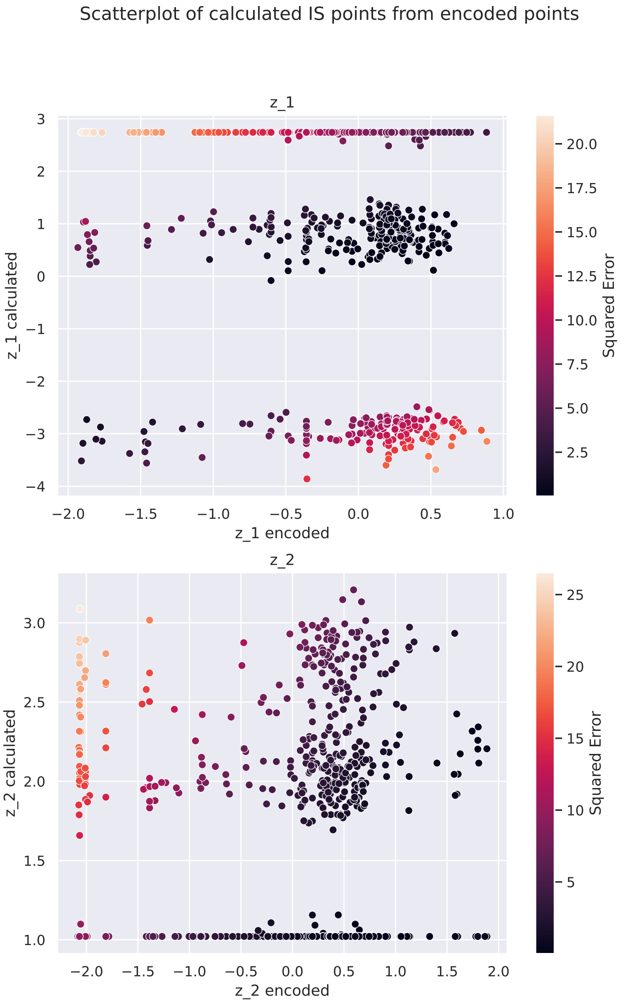

FIGURE 5.13 – Scatterplot of experiment 3 encoder

6 Conclusion

In this chapter we present the conclusions of this work and some ideas for further work.

6.1 Conclusions

We conclude this work by noting that the explored model did not function as expected. The generated data after the IS calculation was not in the region we wanted, and the calculated instance space values were not correlated with the original values. This is a sign that the model is not working as intended. We believe that the model proposed of decoder-encoder is not complex enough to learn the instance space.

6.2 Further work

We believe the instance space reduces the amount of data too much for a proper representation to be learned by a fully connected neural network model. We propose that either the instance space be expanded into more dimensions or that the model use incorporates some process of the dimensionality reduction caused by the instance space analysis.

Bibliography

ADAM, S. P.; ALEXANDROPOULOS, S.-A. N.; PARDALOS, P. M.; VRAHATIS, M. N. No free lunch theorem: A review. *In*: _____. **Approximation and Optimization : Algorithms, Complexity and Applications**. Cham: Springer International Publishing, 2019. p. 57–82. ISBN 978-3-030-12767-1. Available at: https://doi.org/10.1007/978-3-030-12767-1_5.

BENGIO, Y.; COURVILLE, A.; VINCENT, P. Representation learning: A review and new perspectives. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 35, n. 8, p. 1798–1828, 2013.

FACELI, K.; LORENA, A. C.; GAMA, J.; CARVALHO, A. C. P. d. L. F. d. **Inteligência artificial: uma abordagem de aprendizado de máquina**. [*S.l.*]: LTC, 2021.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S.l.*]: MIT Press, 2016. `http://www.deeplearningbook.org`.

HORNIK, K.; STINCHCOMBE, M.; WHITE, H. Multilayer feedforward networks are universal approximators. **Neural Networks**, v. 2, n. 5, p. 359–366, 1989. ISSN 0893-6080. Available at: https://www.sciencedirect.com/science/article/pii/0893608089900208.

KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. 2017. Available at: https://doi.org/10.48550/arXiv.1412.6980.

KOLLA, V. R. K. Paws and reflect: A comparative study of deep learning techniques for cat vs dog image classification. **International Journal of Computer Engineering and Technology**, 12 2020. Available at: https://papers.ssrn.com/abstract=4413724.

MCKINNEY, W. pandas: a foundational python library for data analysis and statistics. **Python High Performance Science Computer**, 01 2011.

MUÑOZ, M. A.; VILLANOVA, L.; BAATAR, D.; SMITH-MILES, K. Instance spaces for machine learning classification. **Machine Learning**, v. 107, n. 1, p. 109–147, Jan 2018. ISSN 1573-0565. Available at: https://doi.org/10.1007/s10994-017-5629-5.

PAIVA, P. Y. A.; MORENO, C. C.; SMITH-MILES, K.; VALERIANO, M. G.; LORENA, A. C. Relating instance hardness to classification performance in a dataset: a visual approach. **Machine Learning**, v. 111, n. 8, p. 3085–3123, Aug 2022. ISSN 1573-0565. Available at: https://doi.org/10.1007/s10994-022-06205-9.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L.; DESMAISON, A.; KöPF, A.; YANG, E.; DEVITO, Z.; RAISON, M.; TEJANI, A.; CHILAMKURTHY, S.; STEINER, B.; FANG, L.; BAI, J.; CHINTALA, S. **PyTorch: An Imperative Style, High-Performance Deep Learning Library**. 2019. Available at: https://doi.org/10.48550/arXiv.1912.01703.

PURI, M.; SOLANKI, A.; PADAWER, T.; TIPPARAJU, S. M.; MORENO, W. A.; PATHAK, Y. Introduction to artificial neural network (ann) as a predictive tool for drug design, discovery, delivery, and disposition: Basic concepts and modeling. basic concepts and modeling. **Artificial Neural Network for Drug Design, Delivery and Disposition**, Elsevier Inc., p. 3–13, 2016.

RICE, J. R. The algorithm selection problem. *In*: RUBINOFF, M.; YOVITS, M. C. (Ed.). Elsevier, 1976, (Advances in Computers, v. 15). p. 65–118. Available at: https://www.sciencedirect.com/science/article/pii/S0065245808605203.

SMITH, M. R.; MARTINEZ, T.; GIRAUD-CARRIER, C. An instance level analysis of data complexity. **Machine Learning**, v. 95, p. 225–256, 2014. ISSN 1573-0565. Available at: https://doi.org/10.1007/s10994-013-5422-z.

SMITH-MILES, K.; MUÑOZ, M. A. Instance space analysis for algorithm testing: Methodology and software tools. **ACM Computing Surveys**, ACM New York, NY, v. 55, p. 1–31, 3 2023. ISSN 0360-0300. Available at: https://dl.acm.org/doi/10.1145/3572895.

YUAN, X.; HE, P.; ZHU, Q.; LI, X. Adversarial examples: Attacks and defenses for deep learning. **IEEE Transactions on Neural Networks and Learning Systems**, v. 30, n. 9, p. 2805–2824, 2019.

Appendix A - Instance Hardness

In this appendix, we will show the calculation of each hardness measures shown in table 3.1 extracted from (Paiva *et al.*, 2022).

k-Disagreeing Neighbours $kDN(\mathbf{x}_i)$**:** gives the percentage of the k nearest neighbours of $\mathbf{x}_i$ which do not share its label. It is described in equation A.1

$$kDN(\mathbf{x}_i) = \frac{\#\{\mathbf{x}_j | \mathbf{x}_j \in kNN(\mathbf{x}_i) \land y_j \neq y_i\}}{k} \tag{A.1}$$

Disjunct Class Percentage $DCP(\mathbf{x}_i)$**:** builds a decision tree using the original dataset $\mathcal{D}$ and considers the percentage of instances in the disjunct of $\mathbf{x}_i$ which share the same label as $\mathbf{x}_j$. The disjunct of an example is the leaf node where it is classified by the decision tree. Equation A.2 shows the calculation of the measure.

$$DCP(\mathbf{x}_i) = 1 - \frac{\#\{\mathbf{x}_j | \mathbf{x}_j \in Disjunct(\mathbf{x}_i) \land y_j = y_i\}}{\#\{\mathbf{x}_j | \mathbf{x}_j \in Disjunct(\mathbf{x}_i)\}}, \tag{A.2}$$

where $Disjunct(\mathbf{x}_i)$ represents the set of instances contained in the disjunct where $\mathbf{x}_i$ is placed.

Tree Depth $TD(\mathbf{x}_i)$**:** returns the depth of the leaf node that classifies $\mathbf{x}_i$ in a decision tree DT, normalized by the maximum depth of the tree built from $\mathcal{D}$:

$$TD(\mathbf{x}_i) = \frac{depth_{DT}(\mathbf{x}_i)}{\max(depth_{DT}(\mathbf{x}_j \in \mathcal{D}))}. \tag{A.3}$$

There are two versions of this measure, using pruned and unpruned decision trees.

Class Likelihood $CL(\mathbf{x}_i)$**:** measures the likelihood of $\mathbf{x}_i$ belonging to its class:

$$CL(\mathbf{x}_i) = 1 - P(\mathbf{x}_i | y_i) P(y_i), \tag{A.4}$$

where $P(\mathbf{x}_i|y_i)$ is the likelihood of $\mathbf{x}_i$ belonging to class y_i, measured in $\mathcal{D}$, and $P(y_i)$ is the prior of class y_i.

Class Likelihood Difference $CLD(\mathbf{x}_i)$: takes the difference between the likelihood of $\mathbf{x}_i$ in relation to its class and the maximum likelihood it has to any other class:

$$CLD(\mathbf{x}_i) = \frac{1 - (P(\mathbf{x}_i|y_i) - \max_{y_j \neq y_i}[P(\mathbf{x}_i|y_j)P(y_j)])}{2} \tag{A.5}$$

Fraction of features in overlapping areas $F1(\mathbf{x}_i)$: the percentage of features of the instance $\mathbf{x}_i$ whose values lie in an overlapping region of the classes:

$$F1(\mathbf{x}_i) = \frac{\sum_{j=1}^{m_D} I(x_{ij} > \text{maxmin}(\mathbf{f}_j) \wedge x_{ij} < \text{minmax}(\mathbf{xf}_j))}{m_D}, \tag{A.6}$$

where I is the indicator function, which return 1 if its argument is true and 0 otherwise, $\mathbf{f}_j$ is the j-th feature vector and:

$$\text{minmax}(\mathbf{f}_j) = \min(\max(\mathbf{f}_i^{c_1}), \max(\mathbf{f}_j^{c_2})), \tag{A.7}$$

$$\text{maxmin}(\mathbf{f}_j) = \max(\min(\mathbf{f}_i^{c_1}), \min(\mathbf{f}_j^{c_2})). \tag{A.8}$$

The values $\max(\mathbf{f}_j^{y_i}$ and $\min(\mathbf{f}_j^{y_i}$ are the maximum and minimum values of $\mathbf{f}_j$ in a class $y_i \in c1, c2$.

Fraction of nearby instances of different classes $N1(\mathbf{x}_i)$: in this measure, first a minimum spanning tree (MST) is built from $\mathcal{D}$. In this tree, each instance of the dataset correspond to one vertex and nearby instances are connected according to their distances in the input space in order to obtain a tree of minimal cost concerning the sum of the edges' weights. $N1$ gives the percentage of instances of different classes that $\mathbf{x}_i$ is connected to.

$$N1(\mathbf{x}_i) = \frac{\#\{\mathbf{x}_j|(\mathbf{x}_i, \mathbf{x}_j) \in MST(\mathcal{D}) \wedge y_j \neq y_i\}}{\#\{\mathbf{x}_j|(\mathbf{x}_i, \mathbf{x}_j) \in MST(\mathcal{D})\}} \tag{A.9}$$

Ratio of intra-extra class distances $N2(\mathbf{x}_i)$: first the ratio of the distance of $\mathbf{x}_i$ to the nearest example from its class to the distance it has to the nearest instance from a different class is computed:

$$IntraInter(\mathbf{x}_i) = \frac{d(\mathbf{x}_i, NN(\mathbf{x}_i) \in y_i)}{d(\mathbf{x}_i, NN(\mathbf{x}_i) \in y_j \neq y_i)}, \tag{A.10}$$

where $NN(\mathbf{x}_i)$ represents a nearest neighbour of $\mathbf{x}_i$. Then, $N2(\mathbf{x}_i)$ is computed as:

$$N2(\mathbf{x}_i) = 1 - \frac{1}{IntraInter(\mathbf{x}_i) + 1} \tag{A.11}$$

Local Set Cardinality $LSC(\mathbf{x}_i$: the Local Set (LS) of an instance $\mathbf{x}_i$ is the set of points from $\mathcal{D}$ whose distances to $\mathbf{x}_i$ are smaller than the distance between $\mathbf{x}_i$ and $\mathbf{x}_i$'s nearest neighbour from another class:

$$LS(\mathbf{x}_i) = \{\mathbf{x}_j | d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_i, NN(\mathbf{x}_i) \in y_j \neq y_i)\} \tag{A.12}$$

$$LSC(\mathbf{x}_i) = 1 - \frac{|LS(\mathbf{x}_i)|}{\#\{\mathbf{x}_j | y_j = y_i\}} \tag{A.13}$$

Local Set Radius $LSR(\mathbf{x}_i)$: the normalized radius of the LS of $\mathbf{x}_i$:

$$LSR(\mathbf{x}_i) = 1 - \min\left(1, \frac{d(\mathbf{x}_i, NN(\mathbf{x}_i) \in y_j \neq y_i)}{\max(d(\mathbf{x}_i, \mathbf{x}_j) | y_j = y_i)}\right) \tag{A.14}$$

Usefulness $U(\mathbf{x}_i)$: corresponds to the fraction of instances having $\mathbf{x}_i$ in their local sets:

$$U(\mathbf{x}_i) = 1 - \frac{\#\{\mathbf{x}_j | d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_j, NN(\mathbf{x}_j) \in y_k \neq y_j)\}}{|\mathcal{D}| - 1} \tag{A.15}$$

Harmfulness $H(\mathbf{x}_i)$: number of instances having $\mathbf{x}_i$ as their nearest neighbour of another class:

$$H(\mathbf{x}_i) = \frac{\#\{\mathbf{x}_j | NN(\mathbf{x}_j) \in y_k \neq y_j = \mathbf{x}_i\}}{|\mathcal{D}| - 1} \tag{A.16}$$

All measures are computed using the entire dataset.

Appendix B - Code for this paper

In this appendix, we will explain the codebase created for this project.

B.1 Introduction

The codebase is written in Python 3.10. The code is available at `https://github.com/gbrlbrbs/tg`. It is implemented as a Pyhon library, with a `pyproject.toml` file for installing with `pip`.

B.2 Configuration

The code needs a YAML configuration file for running experiments. Experiments are defined in the `experiments` folder of the repository, with files as examples.

B.3 Encoder and Decoder creation

Using PyTorch, we can define classes that function as neural network models. We instantiate the encoder and decoder models based on parameters from the configuration file. The encoder and decoder models are defined in the `pyhardgen/nn` folder of the repository. We also created a class for an autoencoder, which is instantiated using both the encoder and decoder models.

B.4 Data generation

For data generation, we define a region of the IS where we will sample points. The code that samples these points is defined in the `pyhardgen/datagen.py` file. It needs the decoder model, a 2 × 2 matrix $\begin{bmatrix}\mathbf{z_1} & \mathbf{z_2}\end{bmatrix}$ defining the bounds of the IS, and the names of each variable returned by the decoder. The code then samples points (100, by default) from a uniform distribution in the region and passes them through the decoder model to generate data. It is then returned as a Pandas DataFrame (Mckinney, 2011).

B.5 Training

The training code is defined in the `pyhardgen/train.py` file. It needs the encoder and decoder models, the training data, and the number of epochs to train for. It then trains the models using the Adam optimizer and the mean squared error as the loss criterion.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO TC	2. DATA 30 de junho de 2023	3. DOCUMENTO Nº DCTA/ITA/DM-018/2015	4. Nº DE PÁGINAS 47

5. TÍTULO E SUBTÍTULO:

Use of generative neural networks for instance space codification and generation of data with specific properties

6. AUTOR(ES):

Gabriel Barbosa Martinz

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Machine Learning, Neural networks, Instance Space Analysis, Generative models

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Aprendizagem (inteligência artificial); Redes neurais; Classificações; Algoritmos; Complexidade computacional; Dados; Computação.

10. APRESENTAÇÃO: **(X) Nacional** ( ) **Internacional**

Trabalho de Graduação, ITA, São José dos Campos, 2023. 47 páginas.

11. RESUMO:

One topic of study in Machine Learning is the study of algorithmic performance and which methodologies may be used to assess this performance. A methodology known as Instance Space Analysis has been used to relate predictive performance in classification algorithms to instance hardness (how hard an instance is for an algorithm to classify). The original methodology has been defined with the instance being an entire dataset, but further work has been made to make the instance as fine-grained as an individual observation. In this work we will build upon this methodology and we propose the creation of a generative neural network model to generate new observations for a classification algorithm with predefined hardness properties.

12. GRAU DE SIGILO:

(X) OSTENSIVO ( ) **RESERVADO** ( ) **SECRETO**