

NEURAL DIMENSIONALITY REDUCTION FOR DATA VISUALIZATION

ALISTER MACHADO DOS REIS

Cover: An artistic representation of different types of data converging to a representation obtained by dimensionality reduction using ShaRP, one of the techniques proposed in this work.

Neural Dimensionality Reduction for Data Visualization

Alister Machado dos Reis
PhD Thesis

The research for this dissertation was conducted at the Visualization and Graphics (VIG) group, part of the Department of Information and Computing Sciences (ICS), Faculty of Science, Utrecht University, the Netherlands.

DOI: <https://doi.org/10.33540/3624>

Neural Dimensionality Reduction for Data Visualization

Neurale dimensionaliteitsreductie voor data visualisatie
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de
Universiteit Utrecht
op gezag van de
rector magnificus, prof. dr. ir. W. Hazeleger,
ingevolge het besluit van het College voor Promoties
in het openbaar te verdedigen op

donderdag 9 juli 2026 des ochtends te 10.15 uur

door

Alister Machado dos Reis

geboren op 19 december 1995
te Porto Alegre, Brazilië

Promotor:

Prof. dr. A. C. Telea

Copromotor:

Dr. M. Behrisch

Beoordelingscommissie:

Prof. dr. H.L. Hardman

Prof. dr. A.P.J.M. Siebes

Prof. dr. R.C. Veltkamp

Prof. dr. M. Verleysen

Prof. dr. M. Worring

“É que ninguém caminha sem aprender a caminhar, sem aprender a fazer o caminho caminhando, sem aprender a refazer, a retocar o sonho por causa do qual a gente se pôs a caminhar.”

“After all, nobody walks without learning to walk, without learning to create the path by walking it, without learning to remake, to retouch the dream for whose cause one has set off down the road.”

— Paulo Freire

ABSTRACT

Information is a crucial resource for humankind, as it allows us to achieve previously unimaginable goals. The bottleneck for the development of complex systems in, for example, Machine Learning, has historically been the ability to collect high quality and highly descriptive data characterizing phenomena of interest. While simple phenomena might be easy to characterize by measuring a small set of their attributes, this quickly stops being the case as our objects of study grow in complexity. Adequately modeling complex events requires collecting accordingly complex data, sometimes containing hundreds or thousands of attributes. This is what we call high-dimensional data.

In this thesis, we study the visualization of high-dimensional data through the lens of Dimensionality Reduction, a family of techniques that casts the problem of finding patterns directly in the data to that of inspecting visual patterns in an alternative representation. More specifically, this work is motivated by noticing how several properties from Neural Networks — a class of Machine Learning models — can form useful synergies with Dimensionality Reduction: such models possess high expressive power, can generalize to unseen data, support a variety of data types, and can express useful inductive biases through their architecture and loss functions. We call the interplay of these two kinds of techniques Neural Dimensionality Reduction.

We first devote ourselves to studying how to *approximate* classic Dimensionality Reduction algorithms through the use of Neural Networks. Such an approximation would allow us to build fast and accurate projections of datasets comprised of millions of data points — a threshold at which traditional Dimensionality Reduction methods hit scalability ceilings. Other techniques exist to address this goal, but they have severe limitations that impede their widespread adoption. We lift these limitations, proposing a new approximate projection algorithm.

Further, we are motivated by a curious aspect of Dimensionality Reduction algorithms: They tend to reproduce, sometimes irrespective of the dataset they process, recognizable visual patterns. That is to say, these data-independent patterns are artifacts of the algorithm’s design, which might mislead users into inferences unsupported by the data. We then investigate whether these patterns can be placed under the user’s direct, explicit control, so that their provenance is known ahead of time and no longer a confounder. Our study shows that indeed, with an appropriate network architecture, such a goal is feasible.

Next, we concern ourselves with how the output of Dimensionality Reduction algorithms is evaluated. The standard way to do that is through the use of Projection Quality Metrics: functions that measure the degree of pattern preservation in a projection. We show an important flaw of such metrics: they behave akin to summary statistics, in that they cannot fully identify whether a projection is indeed showing true patterns existing in the data.

Finally, we turn our attention to Explainable AI, a set of techniques aimed at shedding light onto the outputs of a Machine Learning model. Since Dimensionality Reduction is our object of study, we take a visualization-based approach. We propose extensions to a technique called the Decision Boundary Map, without which we argue the visualization is misleading since it over-simplifies the behavior of the visualized model.

Our contributions show, across a wide spectrum of applications, how Neural Networks can power Dimensionality Reduction techniques: Their versatility allows them to be used to approximate, create, evaluate, and invert projections. Perhaps more importantly, their use effectively imbues DR algorithms with novel, useful capabilities, a sample of which is described in this thesis.

SAMENVATTING

Informatie is een cruciale hulpbron voor de mensheid, omdat het ons in staat stelt voorheen onvoorstelbare doelen te bereiken. Het grootste probleem voor de ontwikkeling van complexe systemen, bijvoorbeeld in machine learning, is typisch het verzamelen van hoge kwaliteit, zeer gedetailleerde, gegevens die karakteriseren fenomenen van interesse. Eenvoudige zijn makkelijk te karakteriseren door een kleine set van hun attributen. Dit is niet meer het geval als de datacomplexiteit toeneemt. Het adequaat modelleren van complexe fenomenen vereist het verzamelen van complexe data, die soms honderden of duizenden attributen bevatten. Dit noemen we hoogdimensionale data.

In dit proefschrift bestuderen we de visualisatie van hoogdimensionale data door middel van dimensionaliteitsreductie, technieken die het probleem van het direct vinden van patronen in de gegevens vertaalt naar het inspecteren van visuele patronen in een alternatieve representatie. In het bijzonder is ons werk gemotiveerd door de bepaling dat verschillende eigenschappen van neurale netwerken — een klasse van machine learning-modellen — nuttige synergieën kunnen vormen met dimensionaliteitsreductie: deze modellen hebben een hoog expressief vermogen, kunnen generaliseren naar onbekende gegevens, ondersteunen verschillende datatypes en kunnen nuttige inductieve biases modelleren via hun architectuur en verliesfuncties. We noemen de combinatie tussen deze twee soorten technieken neurale dimensionaliteitsreductie.

We bestuderen eerst hoe klassieke dimensionaliteitsreductiealgoritmen benaderd kunnen worden met neurale netwerken. Zo een benadering zou ons in staat stellen om snelle en nauwkeurige projecties te berekenen van datasets die bestaan uit miljoenen datapunten — een drempel waarbij traditionele dimensionaliteitsreductiemethoden tegen schaalbaarheidslimieten aanlopen. Andere bestaande technieken die dit doel hanteren hebben sterke beperkingen die hun wijdverspreide toepassing belemmeren. Wij verwijderen deze beperkingen door een nieuw benaderend projectiealgoritme voor te stellen.

We exploreren verder een bijzonder aspect van dimensionaliteitsreductiealgoritmen: ze hebben de neiging om, soms onafhankelijk van de dataset die ze gebruiken, herkenbare visuele patronen te creëren. Met andere woorden: Deze dataonafhankelijke patronen zijn artefacten van het ontwerp van het algoritme, die gebruikers kunnen misleiden tot conclusies die niet door de data worden ondersteund. We onderzoeken verder of deze patronen onder directe, expliciete gebruikerscontrole kunnen worden gezet zodat hun herkomst van tevoren bekend is en geen verstorende factor meer biedt.

Ons werk laat zien dat een dergelijk doel haalbaar is met een geschikte netwerkkarchitectuur.

Vervolgens bestuderen we de evaluatie van de output van dimensionaliteitsreductiealgoritmen. De standaardmethode hiervoor is het gebruik van projectiekwaliteitsmetrieken: functies die de mate van patroonbehoud in een projectie meten. We belichten een belangrijk gebrek van dergelijke metrieken: ze werken als samenvattende statistieken, in die zin dat ze niet volledig kunnen vaststellen of een projectie de echte patronen in de data weergeeft.

Ten slotte richten we onze aandacht op Explainable AI, technieken die proberen de output van een machine learning-model uit te leggen. Omdat dimensionaliteitsreductie ons studieobject is, kiezen we voor een visualisatiegebaseerde aanpak. We presenteren uitbreidingen van de Decision Boundary Map techniek, zonder welke de visualisatie volgens ons misleidend is omdat deze het gedrag van het gevisualiseerde model te veel vereenvoudigt.

Onze bijdragen laten zien hoe neurale netwerken, over een breed spectrum aan toepassingen, technieken voor dimensionaliteitsreductie kunnen gebruiken: Hun flexibiliteit laat ze gebruiken voor het benaderen, creëren, evalueren en inverteren van projecties. Wellicht belangrijker: Het gebruik van neurale netwerken voorziet DR-algoritmen effectief van nieuwe nuttige mogelijkheden zoals dit proefschrift laat zien via voorbeelden.

RESUMO

Informação é um recurso crucial para a humanidade, dada sua capacidade de permitir que alcancemos objetivos previamente inimagináveis. A limitação ao desenvolvimento de sistemas complexos de, por exemplo, Aprendizado de Máquina, tem historicamente sido a habilidade de coletar dados de alta qualidade e altamente descritivos de fenômenos de interesse. Enquanto fenômenos simples podem ser facilmente caracterizados medindo uma pequena parcela de seus atributos, este deixa rapidamente de ser o caso à medida em que nossos objetos de estudo crescem em complexidade. A modelagem adequada de eventos complexos requer a coleta de dados correspondentemente complexos, contendo por vezes centenas ou milhares de atributos. A isto, chamamos de dados de alta dimensionalidade.

Nesta dissertação, estudamos a visualização de dados de alta dimensionalidade através de técnicas de Redução de Dimensionalidade. Esta família de algoritmos transforma o problema de descobrir padrões diretamente nos dados no de inspecionar padrões visuais em uma representação alternativa. Mais especificamente, este trabalho é motivado pela observação de que diversas propriedades de Redes Neurais — uma classe de modelos de Aprendizado de Máquina — podem formar sinergias úteis com técnicas de Redução de Dimensionalidade: tais modelos possuem alta expressividade, podem generalizar para novos dados, suportam diversos tipos de dados, e podem expressar vieses indutivos úteis através de sua arquitetura e funções de perda. À interação entre essas duas classes de técnicas, nós damos o nome Redução de Dimensionalidade Neural.

Primeiramente, nos dedicamos a estudar maneiras de *aproximar* algoritmos clássicos de Redução de Dimensionalidade através do uso de Redes Neurais. Estes algoritmos de aproximação nos permitiriam computar projeções de conjuntos de dados contendo milhões de observações de forma rápida e precisa, ao passo que algoritmos clássicos atingem dificuldades de escalabilidade para conjuntos de dados desta magnitude. Outras técnicas existem para atingir este objetivo, porém estas possuem fortes limitações que impedem seu uso em larga escala. Nosso algoritmo proposto remove estas limitações.

Continuando, somos motivados por uma característica curiosa de algoritmos de Redução de Dimensionalidade: Eles tendem a reproduzir, às vezes independentemente dos conjuntos de dados que processam, padrões visuais reconhecíveis. Ou seja, tais padrões visuais são na verdade artefatos provenientes do algoritmo, e podem potencialmente levar usuários a inferir padrões que não estão presentes nos dados. Assim, investigamos se estes padrões visuais podem ser diretamente e explicitamente controla-

dos pelos usuários, de forma que sua proveniência seja conhecida *a priori*, evitando assim possíveis erros na extração de *insights* dos dados. Nosso estudo demonstra que, com uma arquitetura de rede apropriada, é possível obter um método de projeção com esta propriedade.

Nosso próximo objeto de estudo é a forma de avaliação do que é produzido por algoritmos de Redução de Dimensionalidade. Isto é tipicamente feito através do uso de Métricas de Qualidade de Projeção: funções que medem o grau de preservação de padrões em uma projeção. Nós evidenciamos uma importante falha nestas métricas: elas possuem comportamento semelhante ao de estatísticas de síntese, falhando em identificar se uma projeção realmente mostra padrões verdadeiros — isto é, provenientes dos dados — ou não.

Enfim, dedicamos nossa atenção a *Explainable AI* (Inteligência Artificial Explicável, em tradução livre), um conjunto de técnicas que visa explicar os resultados de modelos de Aprendizado de Máquina. Como Redução de Dimensionalidade é o conjunto de técnicas que estudamos, usaremos uma abordagem baseada em visualização. Propomos extensões para uma técnica chamada *Decision Boundary Map* (Mapas de Fronteiras de Decisão, em tradução livre), sem as quais argumentamos que a visualização pode induzir ao erro ao simplificar em demasia o comportamento do modelo visualizado.

Nossas contribuições demonstram, através de um espectro variado de aplicações, como Redes Neurais podem ser o motor de técnicas de Redução de Dimensionalidade. Sua versatilidade permite que sejam usadas para aproximar, criar, avaliar, e inverter projeções. Talvez de forma mais importante, o seu uso efetivamente imbui algoritmos de Redução de Dimensionalidade com novas e úteis capacidades, uma amostra das quais está descrita nesta dissertação.

PUBLICATIONS

This thesis is a result of the following publications:

- **A. Machado**, A. Telea, and M. Behrisch (2023) ShaRP: Shape-regularized multidimensional projections. In *Proc. EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association. **Best paper award.** ★
- Y. Wang, **A. Machado**, and A. C. Telea (2023) Quantitative and Qualitative Comparison of Decision-Map Techniques for Explaining Classification Models. *Algorithms*, 16(19):438.
- A. C. Telea, **A. Machado**, and Y. Wang (2024) Seeing is Learning in High Dimensions: The Synergy Between Dimensionality Reduction and Machine Learning. *SN Computer Science* 5:279, Springer.
- **A. Machado**, M. Behrisch, and A. Telea (2024) Exploring classifiers with differentiable decision boundary maps. In *Computer Graphics Forum* (Vol. 43, No. 3). **Honorable Mention award.** ★
- **A. Machado**, A. Telea, and M. Behrisch (2024) Controlling the scatter-plot shapes of 2D and 3D multidimensional projections. In *Computers & Graphics*, 124, 104093.
- **A. Machado**, M. Behrisch, and A. Telea (2025) Necessary but not Sufficient: limitations of projection quality metrics. In *Computer Graphics Forum* (Vol. 44, No. 3).
- **A. Machado**, M. Behrisch, and A. Telea (2025) Extensible TensorFlow implementations of projection quality metrics. In *Proc. EuroVis Workshop on the Gap Between Visualization Research and Visualization Software (VisGap)*.
- **A. Machado** and A. Telea (2026) APPA: A Cluster-Preserving Approximating Parametric Projection Algorithm. In *Proc. 21st International Conference on Computer Graphics, Interaction, Visualization Theory and Applications (GRIVAPP)*. **Nominated for Best Paper award.** ★

The relation between the aforementioned papers and the chapters of this thesis is explained in a footnote at the beginning of every chapter which is based on these publications.

CONTENTS

1	INTRODUCTION	1
1.1	Dimensionality Reduction	2
1.2	Neural Networks	5
1.3	Neural Dimensionality Reduction	6
1.4	Decision Boundary Maps	7
1.5	Contributions	9
1.5.1	Challenges and research questions	9
1.5.2	Structure of this thesis	11
1.5.3	Contribution statement	13
2	BACKGROUND AND RELATED WORK	15
2.1	Introduction	15
2.2	Notations and Preliminaries	15
2.3	Dimensionality Reduction	16
2.3.1	Global <i>vs</i> local DR	18
2.3.2	Linear <i>vs</i> non-linear DR	20
2.3.3	Parametric <i>vs</i> non-parametric DR	21
2.4	Projection Quality Metrics	24
2.4.1	Types of projection quality metrics	25
2.4.2	Metric definitions	26
2.5	Interpreting Visual Patterns in Projections	30
2.6	Inverse Projection and Decision Boundary Maps	33
2.7	Machine Learning	36
2.7.1	Classification	36
2.7.2	Regression	37
2.7.3	Regularization	38
2.7.4	Deep learning for dimensionality reduction	38
3	FAST AND ACCURATE APPROXIMATE PROJECTIONS	43
3.1	Introduction	43
3.2	Related Work	44
3.3	Method	46
3.3.1	Neural Network Projection (NNP)	47
3.3.2	APPA: avoiding low-density areas	47
3.3.3	Sampled APPA	49
3.4	Evaluation	50
3.4.1	Generating projections	51
3.4.2	Projection quality metrics	52
3.4.3	Controlling projection sharpening	53

3.4.4	Projecting unseen data	56
3.4.5	Computational scalability	56
3.5	Discussion and Conclusion	58
4	PROJECTION WITH CONTROLLABLE VISUAL SIGNATURE	61
4.1	Preliminaries	62
4.1.1	Notations	62
4.1.2	Controlling visual signatures	63
4.2	ShaRP: Shape-Regularized Neural Projection	66
4.2.1	Method description	67
4.2.2	Controlling cluster shapes	69
4.2.3	Projection on non-Euclidean spaces	74
4.3	Evaluation	76
4.3.1	Generating shape-regularized projections	77
4.3.2	Measuring projection quality	80
4.3.3	Control of shape regularization intensity	83
4.3.4	Faithfulness to data class structure	83
4.3.5	Generating decision boundary maps	85
4.3.6	Computational performance	88
4.4	Discussion	89
4.5	Conclusion	91
5	LIMITATIONS OF PROJECTION QUALITY METRICS	95
5.1	Introduction	95
5.2	Background	96
5.2.1	What is a good projection?	96
5.2.2	How to construct a good projection?	99
5.3	Fooling Projection Quality Metrics	100
5.3.1	Pattern-destructive phase	101
5.3.2	Pattern-constructive phase	103
5.4	Evaluation	105
5.4.1	Experiment setup	106
5.4.2	Studying the created projections	108
5.4.3	Correlations in fooling behavior	111
5.4.4	Ease of fooling	112
5.4.5	Defining a set of guardrail metrics for projection quality	114
5.4.6	Performance	115
5.5	Discussion	116
5.6	Conclusions and Future Work	118
6	UNDERSTANDING CLASSIFIERS WITH ENHANCED DECISION MAPS	121
6.1	Introduction	121

6.2	Preliminaries	123
6.2.1	Gradient-based explanations for classifiers	124
6.2.2	Limitations of decision maps	125
6.3	Differentiable Decision Boundary Maps	126
6.3.1	Classifier confidence	126
6.3.2	Distance to decision boundary	127
6.3.3	Distance to training data	128
6.3.4	Projected space expansion	131
6.3.5	Sensitivity to mislabeled samples	132
6.3.6	Class variability in 2D space	134
6.4	Case Study: Data Poisoning	135
6.4.1	Preparation: implicit projection	136
6.4.2	The attack setting	137
6.4.3	Carrying out the attack	139
6.4.4	Defending using ∂ DBMs	141
6.5	Discussion	141
6.6	Conclusion	142
7	CONCLUSIONS	145
7.1	Future Work	148
A	APPENDIX	151
A.1	Approximating Parametric Projection Algorithm	151
A.2	Shape-Regularized Projection	156
A.3	Fooling Projection Quality Metrics	166
A.4	Differentiable DBMs	176

INTRODUCTION

Things change, change. You cannot have anything. And least of all can you have the present, unless you accept with it the past and the future.

The Dispossessed
URSULA K. LE GUIN

The ability to collect comprehensive and high-quality data is crucial for human advancement — be it in terms of scientific development or increasing efficiency in organization and decision-making. Whenever humans are able to collect and make sense of more high-quality data, it enables leaps in what can be achieved by us as a collective, such as rapidly developing a vaccine with cutting-edge technology, as was the case during the recent COVID-19 pandemic. This is true across many fields: in sociological studies, data-derived models can be used to understand opinions and their variations over time. In chemistry, such models can be used to predict properties of molecules, leading to higher-quality research into the development of new medication. Models of medical images (acquired by microscopy, ultrasound, CT, or MRI) can be used for diagnosis and prognosis of a wide range of medical conditions.

The data we collect is increasingly complex and of growing cardinality. In other words: we are able to measure more *attributes* — also called *dimensions* or *(independent) variables* — of a specific phenomenon of interest, while at the same time being able to sample said phenomenon more *densely*, producing a high number of *observations* — also called *data points* or *samples*. However, as the size — measured in the number of samples or number of dimensions — of the data increases, so does the difficulty in analyzing it to extract so-called *actionable insights*. That is to say, we wish to obtain information that provides us with knowledge on the phenomena that generated the data and, in turn, ways to take decisions concerning such phenomena. More specifically, we should note that the increase of the dimension count is significantly more challenging (for analysis purposes) than the increase of the sample count. As such, the analysis of so-called *high-dimensional data* — *i.e.*, datasets having tens to hundreds or even thousands of dimensions such as those in object recognition or text classification —, has emerged as a distinct research field at the crossroads of multiple disciplines such

as data science, machine learning, and information visualization (Hoffman and Grinstein, 2002; Yin, 2007; Espadoto et al., 2021b).

High-dimensional data can be analyzed using a wide spectrum of methods. At one end, automated methods coming from machine learning (ML) provide highly expressive ways to detect complex patterns in large data collections in a (near) automated way and with high reliability. However, fully automated methods are not directly applicable in contexts where one has limited information about the nature, frequency, and variability of the patterns to look for — and even less so when one aims to discover previously unknown insights captured by the data. At the other end, fully manual methods coming from information visualization (InfoVis) allow analysts to select and display any desirable subset, or aspect, of a dataset, in custom ways. However, such methods do not scale well for large datasets (especially high-dimensional) in terms of both computational and human effort. In the last two decades, research has aimed to combine the advantages of automatic and manual data exploration, leading to the advent of the field known as *visual analytics* (Keim et al., 2008; Cui, 2019). Visual analytics (VA) techniques inherit the scalability and automation of ML and data mining techniques and, in the same time, empower users to iteratively and interactively steer the exploration process, known also under the name of ‘sensemaking loop’, to create data displays that help extracting increasingly more relevant and valuable answers and knowledge from a given data collection.

1.1 DIMENSIONALITY REDUCTION

The data-generating regime of focus in this work is that of *high-dimensional data*, as explained above. How exactly should we go about exploring a dataset of such kind? What types of tools should we use, and what are their limitations?

Some tools used for analysis of *low-dimensional data* can, in theory, be naively applied to the high-dimensional case. For example, computing descriptive statistics — means, medians, counts — may be helpful for understanding a given dataset. Nevertheless, there is a limitation to their use: since the ultimate goal is to understand the data, we need to be able to elicit which patterns of interest are at play in the data-generating phenomenon. For this goal, using a set of descriptive statistics that grows linearly — *e.g.*, mean, median — or faster — *e.g.*, covariance — with the number of observed attributes quickly becomes unwieldy and ineffective. This calls for some sort of *modeling*, in order to enable the detection and exposition of complex patterns in (equivalently) complex data. Alternatively, several visualization techniques exist for multidimensional data, including parallel coordinate plots (PCPs, Inselberg, 2018), scatterplot matrices (Elmqvist et al., 2008), and table lenses (Rao and Card, 1994; Telea, 2006). Yet, all such

techniques scale well only up to a handful (typically under ten) dimensions.

A simple first approach to modeling is to perform *clustering*. This consists in algorithmically detecting *clusters* — *i.e.*, natural groupings in the data — and then analyzing prototypical members of each cluster as a way to make sense of the data. Algorithms such as *k*-means or Gaussian Mixture Model (GMM) fitting, are example tools often used for this. When dealing with high-dimensional data, however, the application of clustering is not trivial for at least two reasons. The first one is *topological*: It is known that in high-dimensional data space, most of the volume of a cluster lies close to its boundary — this is the so-called *curse of dimensionality*. This makes clustering algorithms start to break down as the number of data dimensions increases. The second reason is *computational*: The amount of time required to fit such a model does not always scale well with the number of dimensions. For example, a flexible GMM fitting procedure would need to compute and update covariance matrices which have size proportional to the square of the number of dimensions. In practice, modeling such data must rely on some assumption that allows low-dimensional encoding of the high-dimensional observed phenomenon.

On top of the difficulties mentioned above, it is often the case that a practitioner — scientist, analyst, or domain expert — must *explore* the data in an interactive fashion, building a mental model of the underlying phenomenon as they go. This advocates for combining data modeling, data visualization techniques, and interaction along the visual analytics paradigm introduced earlier on, with all the challenges mentioned above. In turn, this may preclude the use of heavyweight models which, typically, must rely on a set of priors or assumptions about the data being explored — *e.g.*, architectures tailored for specific data types or reliable label information. In contrast, simpler, more generic, and faster-to-compute models are preferred for such interactive user-driven exploration scenarios, such as ‘throwaway’ models that quickly compute a useful, even if imperfect, representation of data.

An approach that (at least partially) sidesteps the problems mentioned above while lending itself nicely to interactive data exploration is that of *Dimensionality Reduction* (DR, also called Projection). In this category of approaches, a low-dimensional representation of the data is explicitly constructed. This is then visualized — typically as a 2D or 3D scatterplot — allowing a user to see all the collected data at a glance. Additional information about the data can then be encoded in any of the usual visual channels — *e.g.*, marker size, color, or shape. In contrast to all earlier-mentioned visualization techniques, DR methods can handle datasets having tens up to thousands of dimensions and hundreds of thousands up to millions of samples. Even though DR can be used for things other than visualization —

e.g., an intermediate step in a pipeline — we dedicate ourselves to studying Dimensionality Reduction through its use for visualization in this thesis.

How exactly to build such a low-dimensional representation in a useful way — *i.e.*, mapping interesting data patterns to understandable visual patterns — has been an active field of study for several years (Carreira-Perpinán, 1997; Yin, 2007; Espadoto et al., 2021b). Likely the most well-known, and still most-used, technique for DR is Principal Component Analysis (PCA, Jolliffe, 2002). PCA works by finding an orthonormal basis of vectors in the data space such that projection onto this basis preserves as much of the variance from the data as possible. This is equivalent to finding a linear transformation of the data to a low-dimensional space such that a reconstruction error is minimized. The key advantages of PCA, which contribute largely to its still widespread use, include a simple and computationally scalable implementation, deterministic behavior, and a relatively straightforward way to interpret the resulting low-dimensional scatterplots. However, these advantages are often offset by the fact that PCA cannot accurately capture patterns — see Section 2.3.1 for a detailed definition — which occur in datasets having a so-called intrinsic dimensionality higher than two.

Over the years, a plethora of DR techniques has emerged with a varied set of characteristics. While some aim to preserve global aspects of the data, such as (scaled) pairwise distances — this is the case of MDS (Torgerson, 1952), for example. Other DR methods are tailored to preserve local data neighborhoods — such as UMAP (McInnes and Healy, 2018) and t-SNE (van der Maaten and Hinton, 2008). The latter class of neighborhood-preserving projections are of special interest to this work. Firstly, they align well with how a user perceives a projection, *i.e.*, in terms of groupings and neighborhoods of data samples. Projection techniques which create scatterplots that exhibit this point grouping (assuming, of course, that such a grouping does exist in the data) are easier to interpret as they offer a direct way for an analyst to reason about which sample groups exist in the input and, further, explore which attributes are instrumental in determining the appearance of such groups. Moreover, visually exploring such grouping helps users in the construction and refinement of ML models that aim to predict (classify) samples as belonging to several classes (Rauber et al., 2017; Nonato and Aupetit, 2018; Benato et al., 2024a). Separately, neighborhood-preserving projections are, in general, non-linear and local methods, meaning they are more flexible in the kind of projection they can output than linear and global methods such as PCA. This flexibility, in turn, allows a more faithful representation of what the true phenomenon might “look like” in the high-dimensional space.

Yet, such non-linear and local projection techniques come with their own limitations. The time necessary to generate a t-SNE projection, for example, scales quadratically with the number of samples m in the dataset.

Many such techniques also use some form of stochastic optimization which makes them non-deterministic — that is, given the same dataset D , the projection $P(D)$ computed from it may vary for every single run of the same method. To complicate matters further, many projection techniques are not parametric — that is, one cannot assume that the same data sample will get projected at the same location; in turn, this makes such techniques hard to adapt to project changing datasets. In particular, this makes such techniques not *out of sample* — *i.e.*, they cannot project additional samples atop an existing projection without changing said projection as well. Additionally, an analyst might be interested in what other data points are *plausible* but did not show up in the collected dataset. Putting it simply: Assuming that a dataset D projects to a scatterplot $P(D)$, one may wonder which would be the values of the samples that project in the *white spaces* (or empty space) between the scatterplot points. This task corresponds to synthesizing new points from the data distribution — *e.g.*, creating new human face images. Support for this task is present in only a few projection techniques. Last but not least, as mentioned before, many DR techniques are offered to the practitioner. Finding out which technique is suitable for a given use case, and how to optimally fine-tune its hyperparameters, is a time consuming task (Nonato and Aupetit, 2018; Espadoto et al., 2021b).

In summary, we notice a necessity of using some sort of modeling to explore and understand high-dimensional data. We discuss above how such models can easily fall prey to the curse of dimensionality, prompting the use of techniques which sidestep this issue. This thesis focuses on the use of Dimensionality Reduction techniques, specifically those higher in expressive power, which are non-linear and neighborhood-based. We identify several issues with such techniques, such as high computational cost, lack of out of sample support, and instability due to their non-parametric nature. We propose solutions to these shortcomings by using Neural Networks, a particular class of Machine Learning models, which we next describe.

1.2 NEURAL NETWORKS

As already mentioned in Sec. 1.1, modeling phenomena based on observations thereof is of utmost importance for several fields across the sciences.

Building models can take on several forms. Simple models include linear regression models, latent factor analysis, Gaussian Mixture Model fitting, and logistic regression. These models all aim to capture some data-generating distribution — *i.e.*, the phenomenon of interest — by making different sets of assumptions. Their parameters are estimated from data, therefore they are classified as Machine Learning (ML) techniques. In detail, such algorithms are coded not to provide a direct answer, but instead to tell the computer how to build a model from available data as to min-

imize the so-called expected risk. This framework is known as Empirical Risk Minimization (ERM, [Vapnik, 1991](#)).

In recent years, the complexity of the models we are able to create has qualitatively increased with the growing availability of high-performance computing and, in parallel, of large datasets on which to train. A particular class of such models falls under the *Deep Learning* denomination. Deep Learning models work by combining basic units called Artificial Neurons that compute a weighted sum of their inputs, followed by a non-linear transform. Models composed of Artificial Neurons are also called Artificial Neural Networks, or simply Neural Networks (NNs, [Rosenblatt, 1958](#); [Amari, 1967](#)).

NNs have enjoyed widespread adoption across a variety of fields due to their high expressive power — that is, ability to model complex phenomena described by many dimensions or parameters. Training such neural networks is facilitated by powerful libraries taking advantage of highly-parallel hardware such as Graphical Processing Units (GPUs) and Tensor Processing Units (TPUs). Neural Networks are quite versatile, lending themselves well to the creation of models for classification (where the output is a discrete label), regression (where the output is a scalar value), and also for image/audio segmentation, object recognition, and sentence generation, among others.

Given all the above, neural networks have become a key tool in the toolset of the visual analytics practitioner. Separately, the data that such networks generate (infer) has become, by itself, a candidate for visual analytics methods that aim to help ML engineers in designing better predictor models.

1.3 NEURAL DIMENSIONALITY REDUCTION

Neural Networks are excellent function approximators. Indeed, the class of Neural Networks with one hidden layer enjoys a Universal Approximation Theorem stating that, given a sufficient number of neurons, a single-hidden layer NN can approximate any given function ([Hornik et al., 1989](#)). In practice, ML applications typically resort to multi-layer neural networks. While these might not *perfectly* approximate a function of interest, they nonetheless produce a highly-accurate approximation at a reasonable compute budget.

Dimensionality Reduction algorithms can be interpreted as functions mapping points in a high-dimensional (data) space to points in a low-dimensional (projection) space. It stands to reason, then, that they may be approximated by Neural Networks. In fact, Dimensionality Reduction can be cast as a type of *regression* problem from high- to low-dimensional spaces. This approach, originally proposed by the well-known (unsupervised) *autoencoder* class of techniques ([Hinton and Salakhutdinov, 2006](#)),

has been recently specialized by the Neural Network Projection (NNP) technique and its variations (Espadoto et al., 2019a, 2021a; Modrakowski et al., 2022; Oliveira et al., 2023a) to handle dimensionality reduction for visualization purposes.

NNP-class methods — which we next refer to under the generic name of neural DR — have several key advantages for the task of dimensionality reduction. They can mimic any desired projection function, *e.g.*, PCA, t-SNE, or UMAP in a generic way. Firstly, they use relatively simple models consisting of a few fully connected layers and, as such, are simple and fast to train. Extensive experiments (Espadoto et al., 2019a; Kim et al., 2022a) have shown that such models can be trained using only a few thousand samples (high-dimensional data points and their corresponding projections obtained by the user-selected projection technique to learn). Secondly, training time for these models becomes negligible once datasets become large — around 50 thousand samples — when compared to using a full-blown projection technique for the entire dataset (Espadoto et al., 2019a). Once trained, the inference cost is linear in the number of projected samples and dimensions, which makes such models typically much faster than the projection techniques they are learned from. Next, inference works fully parameter-free, meaning users are not bothered by having to experiment with setting often complex hyperparameters, as is the case with many existing projection techniques. Additionally, such methods are deterministic at inference time by construction (the same data sample gets projected to the same location), robust to small perturbations of the input data, and have the out-of-sample ability mentioned in Sec. 1.1. Last but not least, NNP-class methods can be adapted relatively easily to construct inverse projections that map locations in the projection space to plausible data samples that would project there (Espadoto et al., 2019b, 2023a).

1.4 DECISION BOUNDARY MAPS

In the context of data visualization, DR algorithms create (typically) *sparse* representations of high-dimensional phenomena. That is, given a dataset D having a finite number of m elements, $|D| = m$, they create a scatterplot $P(D)$ having the same number of points. As mentioned earlier in Sec. 1.1, such visual representations leave large parts of the projection space — typically the 2D screen on which $P(D)$ is visualized — empty. The analyst does not know what to infer for such points; all that one can do, and what one typically does, is to visually *extrapolate* the data values represented by points in $P(D)$ to small neighborhoods close to those points. However, doing this without explicit help from a tool is risky especially when the gaps between neighbor points in $P(D)$ are large. Without assistance, we really do not know what kind of data would project in these gaps.

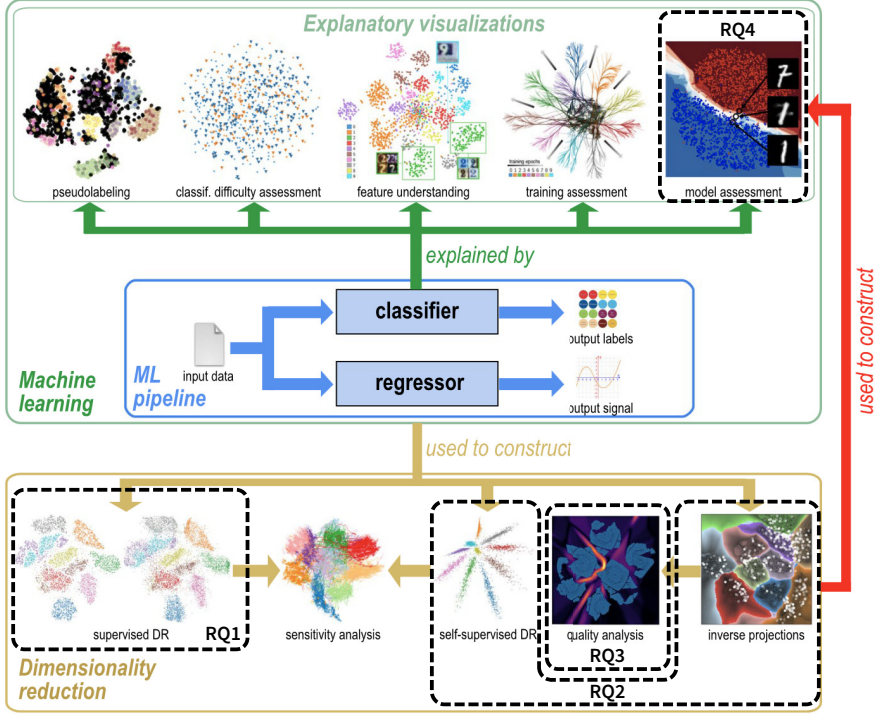


Figure 1.1: Two-way interaction between machine learning (ML) and dimensionality reduction (DR) workflows. ML algorithms can be used to construct DR techniques. In turn, these can be used to construct explanatory visualizations for ML. Our research contributions (RQ1-4, see Section 1.5) are marked atop of this workflow. Figure adapted from (Telea et al., 2024).

Inverse projection techniques fill in such gaps. Given a dataset D and its projection $P(D)$ created by some user-chosen technique P , an inverse projection constructs a smooth extrapolation of the mapping that (approximately) undoes P over the entire projection space (dos Santos Amorim et al., 2012; Espadoto et al., 2019b; Blumberg et al., 2025). Among other applications, inverse projections allow the construction of so-called *decision boundary maps* (DBMs) which are dense visual representations of the behavior of a trained classification model over a dataset. This allows for the at-a-glance visualization of regions where the classifier consistently outputs a given decision (with inferred classes encoded as colors in the projection space), but also the investigation of boundaries between those zones — hence the technique’s name — shedding light on possible causes for the shift in a classifier’s output (Rodrigues et al., 2018, 2019; Oliveira et al., 2022).

1.5 CONTRIBUTIONS

Dimensionality reduction, originally developed in the context of statistics and machine learning (Jolliffe, 2002; Hinton and Salakhutdinov, 2006), has been further refined in the field of information visualization and visual analytics to yield several techniques covering numerous data exploration applications (Nonato and Aupetit, 2018). In parallel, machine learning has known an at least as prolific development with many techniques covering the increasingly accurate prediction of increasingly more complex phenomena. Recent developments have highlighted that such advances can exploit *synergies* in the two fields (see Fig. 1.1): On the one hand, ML techniques can be used to create better — e.g., faster, easier to use, out-of-sample, and more robust — DR algorithms for data visualization. This approach is suitably dubbed ‘learning for seeing’ (Telea et al., 2024). On the other hand, these algorithms can be used to create various visualizations that explain the working of ML models — an approach which is dubbed ‘seeing for learning’ (Telea et al., 2024). Chapter 2 elaborates on these aspects by presenting related work at the crossroads of dimensionality reduction and machine learning which is relevant to our work.

However, within the scope the aforementioned synergy, several limitations are apparent to both the use of ML for constructing projections and the use of projections to create effective visualizations for ML model explanation. We next highlight these limitations which our work aims to address.

1.5.1 *Challenges and research questions*

Approximation: While neural projections, such as NNP and its variants, score well on most desirable requirements for a projection technique (see Sec. 1.3), they all have one key limitation: Compared to state-of-the-art DR techniques such as t-SNE and UMAP, the produced scatterplots are significantly *fuzzier*. Putting it simply, groups of samples which appear well separated both in data space and in projections constructed by techniques such as t-SNE and UMAP appear unnecessarily mixed up when being projected with neural methods. This significantly decreases the perceived advantage of such neural methods as opposed to classic ones such as t-SNE or UMAP. As noted in earlier studies on NNP, this fuzziness is not entirely unexpected given the trade-off between overfitting and ability to generalize present in virtually any training process of a ML model (Espadoto et al., 2019a). Yet, we still pose the question:

RQ1: Can we create neural projections having all desirable properties — e.g., genericity, scalability, high quality, soundness — of such state-of-the-art methods while reducing fuzziness?

Pattern control: Existing projection techniques offer various mechanisms to control a wealth of aspects of the resulting scatterplots such as its deterministic generation from input data (van der Maaten, 2009); the out-of-sample ability to add new points to an existing projection scatterplot (Espadoto et al., 2019a); the creation of dynamic projections for time-dependent data (Vernier et al., 2020a); and, most importantly, the preservation of various distance and neighborhood properties defined between the data samples when generating the projection scatterplot (Espadoto et al., 2021b). All these properties help the analyst’s task of interpreting the projection scatterplot, *i.e.*, inferring data-related aspects from the visual elements present in the plot. Yet, to our knowledge, no projection algorithm offers users a way to control the shapes of the *visual patterns* that such plots consist of. Rather, such shapes are implicitly controlled by the nature of the projection technique being used — for example, t-SNE tends to create round, organic, densely-packed cluster patterns, while UMAP creates small, high-density, clusters separated by large amounts of whitespace. We believe that such implicitly created patterns can mislead users in inferring actual data properties and therefore pose the question:

RQ2: Can we create neural projections having all desirable properties of such state-of-the-art methods and, additionally, offering an explicit control of the point patterns they create?

Projection quality: An issue that immediately arises when it is possible to generate many different projections of the same data is *which one should be used*. Ideally, a DR algorithm should preserve *all* of the interesting data patterns by mapping them in perceivable manner to visual patterns in the projection. However, it is not expected that a perfect projection exists, even for reasonably simple datasets. Typically, reducing the number of dimensions incurs distortions, be it in how distances are represented or in how the similarity relations are portrayed (Nonato and Aupetit, 2018).

To assess the degree of preservation of patterns in a projection, so-called Projection Quality Metrics (PQMs) are typically used. Many such metrics exist, for example the degree of pairwise distance distortion present in the projection; the proportion of preserved neighborhood relations; or the preservation of same-label clusters from the data space to the projection space (for labeled data). The higher a projection scores on several such metrics, the better is that projection considered to depict its input data (Espadoto et al., 2021b). There are, however, two main problems with the usage of such quality metrics. First, most projection techniques do not *explicitly* optimize for these metrics in their operation. This is due to a range of reasons including that many such metrics exist (and thus it is hard to optimize for all of them simultaneously); and that several such metrics do not have a smooth, differentiable, formulation, which precludes their di-

rect usage in many optimization algorithms (Nonato and Aupetit, 2018). A second, equally important, issue, regards how one can interpret PQM values, namely:

RQ3: Are high values of projection quality metrics a sufficient indicator that a projection faithfully captures underlying data patterns?

Enhanced decision maps: Our three contributions listed above cover the first part of the pipeline outlined in Fig. 1.1, namely how ML, specifically deep learning, can be used to enhance DR. Our fourth and last contribution, described below, covers the second part of the said pipeline, namely how DR can be used to provide explanations of ML models. Specifically, we focus here on decision boundary maps (DBMs) which, as outlined in Sec. 1.4, are simple but effective tools that depict the behavior of a trained classification model. However, existing DBM techniques show only a limited amount of information at every pixel (which maps one data sample): Typically, the label inferred by the model for that sample and, optionally, the inference’s confidence. We thus ask ourselves the following question:

RQ4: How to extend decision boundary map visualizations in order to expose additional insights regarding the operation of the visualized classification model?

1.5.2 Structure of this thesis

We next describe how we answer each one of the research questions elicited above throughout the work developed in this thesis.

In Chapter 3, we answer RQ1 and propose an Approximating Parametric Projection Algorithm (APPA), an extension of NNP which significantly reduces the aforementioned fuzziness while keeping all earlier-mentioned desirable properties of neural methods. Key to this development is the incorporation of an additional cost term in the loss function of the neural network which is trained to mimic a given projection technique. This additional loss term forces the projected points to avoid low-density areas in the projection space, *i.e.*, favors the appearance of sharp-delimited point clusters. We compare APPA to NNP on a range of datasets and projection techniques to mimic and show that our technique structurally decreases the said fuzziness for a negligible computational cost: training takes double the time taken by NNP, while inference incurs no cost penalties.

In Chapter 4, we address RQ2 by proposing a specific neural projection that allows *direct, explicit* control of visual patterns. Our investigation starts by noticing that DR algorithms produce consistent patterns *across* datasets. We name these patterns an algorithm’s *visual signature* and study how to control them. To this end, our proposed neural technique, called Shape Regularized Projection (ShaRP), innovates by using

Variational Auto-Encoders (VAE), to incorporate specific shape constraints specified by the user. ShaRP allows one to factor out the uncontrolled generation of various pattern shapes, present in earlier DR techniques while keeping the projection quality close to values delivered by state-of-the-art techniques. We also exploit ShaRP’s structure to create projections to non-Euclidean spaces, for example the surface of a sphere ($\mathcal{S}^2$), where more similarity relationships can be represented than in Euclidean space due to the periodicity of the sphere’s structure.

In Chapter 5, we answer RQ3 and posit that PQMs behave as summary statistics: They do not *identify* the objects they measure but, rather, they allow for enough variability to enable poor projections to receive high quality scores. In other words, we hypothesize that high values of Projection Quality Metrics correspond to *families* of projections for any given dataset; within such a family, there are degenerate projections that do not preserve any data patterns. We provide extensive evidence to the truth of this hypothesis by building a neural network-based pipeline to automatically generate a wide range of projections, containing both random and structured visual patterns, that are nonetheless deemed high-quality by established PQMs. To train this network, we propose a way to continuously approximate any PQM formulation (be it a smooth function or not) using a separate neural network. We thus address the first problem mentioned above in our discussion on quality metrics. Our work shows that PQMs are necessary, but not sufficient, indicators of a projection’s quality — that is, low values of such metrics indicate that one has a poor projection which should not be used to analyze the projected data; however, high values of such metrics are not sufficient to indicate that that projection captures well the data structure.

In Chapter 6, we address RQ4 by exploring the standard DBM visualizations. We show that the depicted information by such DBMs can be actually misleading in terms of understanding the operation of the explored classification model — for example, regarding the model’s ability to correctly generalize its output classes to unseen data. We address this issue by proposing several visualization extensions that augment standard DBMs by extra information that depicts where the model is poorly supported, where the DBM visualization distorts data-space distances, where the model training is sensitive (thus, brittle), and why the model’s confidence varies across the data space. We illustrate our enhanced DBM visualizations for the exploration of a data poisoning application, thereby showing its potential added value in practical ML engineering.

Chapter 7 concludes this thesis by revisiting all the individual contributions of this dissertation (RQ1-RQ4). We discuss the learnings achieved during the development of each technique that we propose to address RQ1-RQ4, highlight our main results in answering these research questions, and also point to limitations and directions for future work.

1.5.3 Contribution statement

This thesis has resulted in the following contributions:

- In (Machado and Telea, 2026), we published the Approximating Parametric Projection Algorithm (APPA) and its sample-based version SAPPa. Both are implemented in a public software package available at <https://github.com/amreis/appa>. This work’s relevance, novelty, and significant improvement over the state-of-the-art have resulted in a nomination for the **Best Paper Award** at GRIVAPP 2026.
- In (Machado et al., 2024e, 2023), we described ShaRP, the Shape-Regularized Neural Projection algorithm, whose code is also publicly available at <https://github.com/amreis/ShaRP>. The novelty of the algorithm has motivated it being awarded the **Best Paper** award at EuroVis Workshop on Visual Analytics (EuroVA) in 2022.
- In (Machado et al., 2025b), our evaluation of Projection Quality Metrics provided, **for the first time**, substantial evidence to their strong limitations in identifying high-quality projections. The code for the experiments and the results are both made publicly available at <https://github.com/amreis/fooling-projection-metrics> and <https://amreis.github.io/fool-proj-metrics>, respectively.
- The work above also resulted in the development of a **public and extensible** implementation (Machado et al., 2025a) of 16 Projection Quality Metrics (<https://github.com/amreis/tf-projection-qm>), available as a Python package at <https://pypi.org/p/tensorflow-projection-qm>.
- In (Machado et al., 2024d), we proposed augmentations to the Decision Boundary Map technique, with code publicly available (<https://github.com/amreis/differentiable-dbm>). The quality of the visual explanations produced by our technique, as well as its relevance to the field of Explainable AI, motivated it receiving the **Honorable Mention** award at EuroVis 2025.
- Again in (Machado et al., 2024d), we developed a fast implementation of DeepFool, an algorithm for adversarial example generation, available publicly in the same repository. This effectively **solved an engineering problem** by improving the scalability of the technique, enabling the work we developed.

BACKGROUND AND RELATED WORK

Ninguém educa ninguém, ninguém educa a si mesmo, os homens se educam entre si, mediatizados pelo mundo.[†]

Pedagogia do Oprimido
PAULO FREIRE

2.1 INTRODUCTION

As outlined in Chapter 1, the focus of this thesis is at the crossroads of machine learning (more specifically, deep learning) and dimensionality reduction. Our contributions, listed in Sec. 1.5 and detailed in the next chapters, will cover both using deep learning for enhancing dimensionality reduction — what we earlier called ‘learning for seeing’ — and using dimensionality reduction (DR) to produce enhanced explanations of deep learning models — what we earlier called ‘seeing for learning’.

In this chapter, we introduce the technical background needed to anchor our further investigations in ‘learning for seeing’ and ‘seeing for learning’. In Section 2.2, we introduce the general terms and notations that we will use throughout this thesis. Section 2.3 presents dimensionality reduction including an overview of the key techniques used in this area. Section 2.4 introduces the main quality metrics used in DR literature to gauge the quality of computed projections. Section 2.7 gives a (necessary limited) overview of the key machine learning (ML) concepts, techniques, and tools which are relevant to our work, with a focus on neural network approaches.

2.2 NOTATIONS AND PRELIMINARIES

Throughout this work, we will use the following notation for our objects of study. Along this general notation, we will introduce additional specific notation as needed in each of the following chapters.

[†] “No one teaches another, nor is anyone self-taught. People teach each other, mediated by the world.”

We use upper case bold-face letters (e.g., $\mathbf{X}, \mathbf{Y}$) to denote matrices, lower case bold-face letters (e.g., $\mathbf{x}, \boldsymbol{\mu}$) to denote vectors, and regular letters (e.g., $x_{ij}, \mu, \bar{x}$) to denote scalars, respectively.

We denote the Euclidean norm of a vector by $\|\mathbf{x}\|$ or $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^T \mathbf{x}} = \sqrt{\sum_i x_i^2}$, its L1 norm by $\|\mathbf{x}\|_1 = \sum_i |x_i|$, the Frobenius norm of a matrix by $\|\mathbf{X}\|_{\text{Fro}} = \sqrt{\sum_{i,j} x_{ij}^2}$ and the expected value of a function of a random variable $\mathbf{z}$ distributed according to probability density function p by $\mathbb{E}_{\mathbf{z} \sim p}[f(\mathbf{z})]$, respectively.

A dataset $\mathcal{D}$ is a finite set of m indexable *observations* $\mathbf{x}_i$, $1 \leq i \leq m$, also called *data points*, *samples*, or *examples*. Each $\mathbf{x}_i$ is a *vector* with n dimensions, i.e., $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in \mathbb{R}^n$. When convenient, we can also represent a dataset $\mathcal{D}$ as its *design matrix* $\mathbf{X} \in \mathbb{R}^{m \times n}$ where we stack each vector $\mathbf{x}_i$ along the rows of $\mathbf{X}$:

$$\mathbf{X} = \begin{array}{c} \overbrace{\begin{bmatrix} - & \mathbf{x}_1^T & - \\ - & \mathbf{x}_2^T & - \\ - & \mathbf{x}_3^T & - \\ & \vdots & \\ - & \mathbf{x}_m^T & - \end{bmatrix}}^{n \text{ dimensions}} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ x_{31} & x_{32} & \dots & x_{3n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} . \end{array}$$

When a dataset possesses ground-truth *label* — i.e., class — information, we refer to the label of a specific data point $\mathbf{x}_i$ as c_i or, alternatively, $C(\mathbf{x}_i)$, where $C : \mathbf{X} \rightarrow [K]$ is a function mapping each data point to its class and $[K] = \{1, \dots, K\}$ is the set of positive integers no greater than K . Note that, strictly speaking, labels in our work can be *categorical*; the mapping of such K categories to the integers in $[K]$ is used purely for notation convenience.

2.3 DIMENSIONALITY REDUCTION

We denote a low-dimensional *projection* $\mathbf{Y}$ of $\mathbf{X}$ by $\mathbf{Y} = P(\mathbf{X})$. We also call P a *dimensionality reduction* (DR) algorithm applied to $\mathbf{X}$. A projection P maps each data point $\mathbf{x} \in \mathcal{D} \subset \mathbb{R}^n$ to $\mathbb{R}^q$, with $q \ll n$. This thesis focuses on the cases where $q \in \{2, 3\}$, giving rise to projections that can be represented and studied as, respectively, 2D and 3D *scatterplots*. Given their one-to-one mapping in the context of this thesis, we overload the $P(\mathbf{X})$ notation to refer, where no ambiguity arises, to both the projection as well as its resulting scatterplot, where the coordinates produced by P encode the positions of the points. We may also refer to the projection of a specific data point $\mathbf{x}$ as $\mathbf{y} = P(\mathbf{x})$; this is a slight abuse of notation, as P is, in general, only meaningful when computed over an entire dataset at once. That is, we cannot — again, in general — model a projection as a function $P : \mathbb{R}^n \rightarrow \mathbb{R}^q$,

as this would imply that one can compute the projection $\mathbf{y} = P(\mathbf{x})$ of a *single*, isolated, sample $\mathbf{x} \in \mathbb{R}^n$, regardless of other samples which, along with $\mathbf{x}$, form the dataset $\mathcal{D}$. There are specific cases when the above is, however, possible, as explained further in the context of *parametric* projections and, also, for *inverse* projections. Throughout our work, we will distinguish these two situations as explained in their specific contexts.

Note that, in the description above, we consider projections as *unsupervised* techniques. That is to say, a projection by default does not ingest any class information, building a low-dimensional representation based solely on similarities among the data points themselves. In Chapter 4, we consider the use of self-supervised techniques, in which label information is derived from the datapoints and subsequently used as input to an algorithm — an approach directly comparable to unsupervised techniques. In the same Chapter, we explore the effects of using ground truth class information, rendering these techniques *supervised*, as a way to evaluate the limits of our proposed technique under the ‘perfect’ information regime.

Typical projection algorithms do not only operate based on their input datasets $\mathcal{D}$ but also under the control of so-called *hyperparameters*. These are used to control various operations of the algorithm *e.g.* how accurately some quality metric, or cost function, is approximated; or the amount of computational effort used to perform its work. When relevant to the discussion, we will denote these hyperparameters by θ and the corresponding parameterized projection algorithm by P_θ , respectively.

In the context of visual analytics applications, users typically study the scatterplot $P(\mathbf{X})$, with points optionally enriched to show a few data dimensions via color, size, shape, or annotations, looking for salient *visual patterns*. Simply put, the key added-value of a (good) projection relies in its ability to capture relevant patterns present in the data $\mathcal{D}$ and expose them so users can efficiently and effectively spot them. We discuss the interpretation of such projection scatterplots and their underlying visual patterns further in Sec. 2.5. Figure 2.1 illustrates the overall operation of projections.

Tens of different DR methods exist, with diverse characteristics and mathematical underpinnings. These are described from technical perspectives in several surveys (Hoffman and Grinstein, 2002; Yin, 2007; van der Maaten and Postma, 2009; Bunte et al., 2012; Sorzano et al., 2014; Cunningham and Ghahramani, 2015; Liu et al., 2015; Heulot et al., 2017; Nonato and Aupetit, 2018; Espadoto et al., 2021b). The differences in these algorithms’ inner workings lead to corresponding differences in the projections they generate. A (necessarily simplified) way to capture such inner workings in a general formulation can be outlined as follows. Consider a dataset $\mathbf{X}$ and its projection scatterplot $P(\mathbf{X})$. As noted above, a good projection should preserve various aspects of the dataset $\mathbf{X}$. This preservation can be modeled — up to various extents — by a *cost function* of the form $Q(\mathbf{X}, P(\mathbf{X})) \in \mathbb{R}_+$

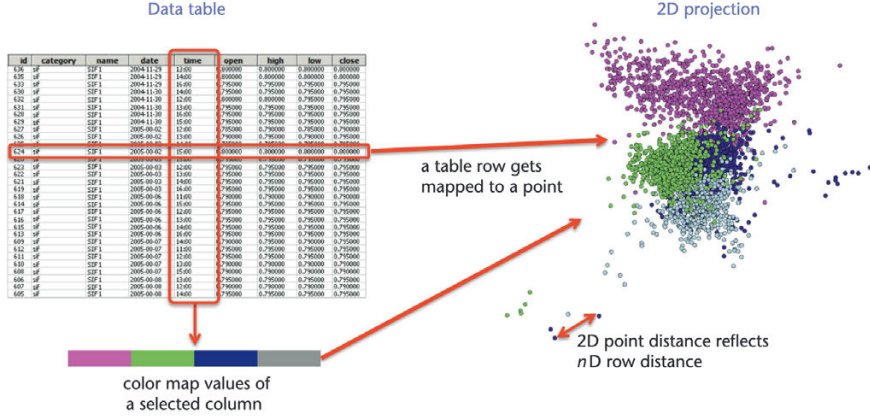


Figure 2.1: Typical operation of projection algorithms. Given a data table which encodes the design matrix X , a projection algorithm creates a scatterplot $P(X)$ that can be explored to reason about data properties — in this example, the distances between scatterplot points are computed to reflect the distances between the data samples. Additional attributes such as color can be used to map one (or a few) dimensions of the data table. Image taken from (da Silva et al., 2016).

— that is, the lower this cost is, the better is said that $P(X)$ preserves, or captures, the desired data patterns from X . Following this, the overall operation of a projection algorithm P can be seen as a *minimization* of the cost function Q subject to a given dataset X .

Following the above, projection algorithms can be categorized in terms of how they approach the said minimization problem. We next present a (necessarily brief) categorization of projection techniques among two technical axes or characteristics, namely *where* the projection algorithm measures the cost Q (Sec. 2.3.1) and *how* the algorithm aims to minimize Q (Sec. 2.3.2). A separate important distinction of projection algorithms based on their parametric behavior (or lack thereof) is discussed in Sec. 2.3.3.

2.3.1 Global vs Local Dimensionality Reduction

Firstly, consider the nature of the optimization carried out by such projection algorithms. It is possible to compute transformations that rely on the *entirety* of a dataset at once, giving equal importance to each data point — such algorithms are called *global*. On the other hand, algorithms might instead focus their optimization efforts on neighborhoods around each data point — we call such techniques *local*. That is, in the global case, one aims to minimize the cost function $Q(X, P(X))$ for the *entire* dataset X ; in con-

trast, the local case, one computes different costs $Q(\mathbf{S}, P(\mathbf{S}))$ for subsets of points $S \in X$ and aims to minimize these costs separately.

Global Dimensionality Reduction comprises algorithms such as Principal Component Analysis (PCA, [Pearson, 1901](#)), arguably the most well-known DR algorithm. PCA works by finding a linear transformation $\mathbf{W} \in \mathbb{R}^{d \times q}$ such that multiplication by $\mathbf{W}$ captures the most *variance* present in the dataset $\mathbf{X}$. Alternatively, PCA can be seen as minimizing the reconstruction error¹ $\|\mathbf{X} - \mathbf{X}\mathbf{W}\mathbf{W}^T\|_{\text{Fro}}$. This is a global objective: a *single* optimal $\mathbf{W}$ exists that reconstructs $\mathbf{X}$ as best as possible in the L2-norm sense.

Another notable global DR algorithm is Multidimensional Scaling (MDS, [Kruskal, 1964](#)). The algorithm computes the positions $\mathbf{Y} = P(\mathbf{X})$ as to minimize a normalized measure of pairwise distance distortion called Stress, defined as

$$\sum_{i=1}^m \sum_{j=i+1}^m (\|\mathbf{x}_i - \mathbf{x}_j\| - \|\mathbf{y}_i - \mathbf{y}_j\|)^2.$$

This measure computes a single scalar for the entire projection taking into account *all* the present “errors” introduced in the projection, making MDS a global DR method.

Isomap ([Tenenbaum et al., 2000](#)) replaces the Euclidean distances in MDS with geodesic distances along an approximation of the manifold containing $\mathbf{X}$. It is also a global projection method. However, due to its use of geodesic distances, Isomap has to use a more complex minimization algorithm than MDS.

Local Dimensionality Reduction is a family of algorithms devoted to preserving local characteristics from the data space, possibly introducing distortions that lead to worse global quality overall. For example, such an algorithm might choose to place data points at distances that do not correspond to their (scaled) original distances in the original data space. This trade-off is justified: users perceive projections in terms of the visual patterns therein — *e.g.*, groupings, collinearity, outliers — more than in terms of numerical accuracy to a given reference metric. Local methods enjoy wider adoption precisely due to this characteristic: they preserve important *qualitative* aspects of the data distribution in its original space. A separate advantage of local DR is that minimizing a given cost on a *smaller* and more *local* subset of points $\mathbf{S} \in \mathbf{X}$ is, in general, simpler (and computationally faster) than performing the same minimization on the entire set $\mathbf{X}$. Indeed, intuitively put: As the set $\mathbf{S}$ becomes smaller and also contains points which are closer to each other in the data space $\mathbb{R}^n$, it is arguably easier to construct a low-dimensional point-set $P(\mathbf{S})$ which preserves whichever quality the cost function Q aims to capture.

¹ We represent application of PCA by right-multiplication by $\mathbf{W}$ due to our convention of representing each element in $\mathbf{X}$ in a row.

Notable local DR algorithms are, for example, t-Stochastic Neighbor Embedding (t-SNE, [van der Maaten and Hinton, 2008](#)), a non-linear method that fits differently-scaled Gaussians to neighborhoods in the data space then maps those Gaussians to Student T distributions in the projection; and Uniform Manifold Approximation and Projection (UMAP, [McInnes and Healy, 2018](#)), an algorithm that approximates geodesic distances in the data space and uses those distances to compute a projection. We will encounter both algorithms several times during our further explorations of projections in this thesis.

2.3.2 Linear vs Non-Linear Dimensionality Reduction

Following the (simplified) explanation of projection algorithms as techniques that aim to minimize some cost function Q , we have several ways in which this minimization can be performed. In this respect, we distinguish two main classes of methods, as follows.

Linear DR algorithms work by linearly transforming the data they work on. This means they can be fully described by a transformation $\mathbf{X} \mapsto \mathbf{X}\mathbf{W}$, where $\mathbf{W}$ is a $n \times q$ matrix. This leads to some advantageous characteristics. First, linear computations are simple to implement and efficient to execute. Secondly, and more importantly in the context of data visualization, the columns of $\mathbf{W}$ can be directly interpreted: in the case of PCA, they are the Principal Components associated with the projection, *i.e.*, they represent linear combinations of features from the original data that, each in turn, preserve the most variance present in the dataset. Exactly how much variance each principal component accounts for is also readily available when using PCA. If the algorithm explicitly builds $\mathbf{W}$, it is directly applicable to unseen data *without any recomputation* — this is a highly useful characteristic which we revisit in the following section. Unfortunately, linear transformations are not powerful enough to produce expressive, useful projections unless the dataset’s intrinsic dimensionality is low ([Espadoto et al., 2021b](#)). Their linearity constrains them and prevents them from creating visual patterns corresponding to more complex data patterns.

Non-Linear methods, on the other hand, are not restricted in the type of transformation they compute. They can freely optimize projected points $\mathbf{y}_i$ ’s positions in order to minimize any criterion Q in order to produce the projection scatterplot. This is by far the larger family of projection methods: Their higher expressive power motivates active research into how to build increasingly sophisticated methods with desirable properties. Simply put, one can design increasingly sophisticated cost criteria Q (which, in turn, model the capturing of increasingly more complex data patterns from $\mathbf{X}$ into $P(\mathbf{X})$) if one allows for more sophisticated non-linear optimization algorithms to address the minimization of Q .

However, being non-linear, the optimization loop of such projection algorithms is usually computationally intensive, hindering their scalability with respect to the number of samples and/or number of dimensions in a dataset $\mathbf{X}$. These algorithms do not typically build an explicit projection function as linear methods do. Instead they optimize projected points' positions directly — *i.e.*, they are non-parametric. This means they cannot directly handle the projection of unseen data, requiring full (and expensive) recomputation (we detail this aspect further in the next section). Nevertheless, these methods enjoy widespread adoption, with t-SNE and UMAP being state-of-the-art techniques in the data scientist's toolbox.

2.3.3 Parametric vs Non-Parametric Dimensionality Reduction

The third and final axis along which we discuss projection techniques regards how these operate with respect to their input dataset $\mathbf{X}$ and the data space $\mathbb{R}^n$ in which $\mathbf{X}$ is embedded. We distinguish here parametric *vs* non-parametric methods, as follows.

Non-parametric methods are those that work *directly* with the projection they generate, modifying the position of each individual $\mathbf{y}_i = P(\mathbf{x}_i)$. They enjoy, therefore, greater flexibility during the optimization of their (often complex) cost functions. A key representative of this family of methods is t-SNE (van der Maaten and Hinton, 2008). This method fits one Gaussian distribution per data point $\mathbf{x}_i$ such that all Gaussians have a user-chosen perplexity. These Gaussians are then used to compute conditional probabilities that each data point $\mathbf{x}_i$ would choose any other $\mathbf{x}_j$ as its neighbor. These probabilities are symmetrized and normalized. A similar procedure is done in the low-dimensional space, using Student t distributions instead of Gaussians to counteract the shrinking of the space itself. The positions of the projected points $\mathbf{y}_i$ are directly optimized by gradient descent on the Kullback-Leibler Divergence between the high-dimensional Gaussians and the low-dimensional Student t distributions. This effectively creates a force-directed placement, where each point in the projection can *individually* move in space to find its best projection spot. The method is successful in showing structure from the data in the projection, but scales quadratically in time and space with the number of samples in the dataset. t-SNE is local, non-linear, and non-parametric.

Another well-known non-parametric method is UMAP (McInnes and Healy, 2018). This method aims to preserve neighborhoods in $\mathbf{X}$ by computing an approximation of the manifold on which $\mathbf{X}$ lies. The approximation is built using a modified weighted k -Nearest Neighbors graph. The k NN graph is then mapped to the projection space by a modified force-directed layout algorithm. Compared to t-SNE, UMAP has a relatively higher computational speed.

Non-parametric methods project a *dataset* $\mathbf{X} \in \mathbb{R}^n$. When this dataset $\mathbf{X}$ changes — even slightly — the resulting $P(\mathbf{X})$ changes as well. Even stronger, non-parametric methods do not guarantee *determinism*: That is, given the same dataset $\mathbf{X}$, computing the projection $P(\mathbf{X})$ can lead to different results for different runs of the same technique P . This is the case, for instance, for methods such as t-SNE which use stochastic optimization to minimize the underlying cost Q , which is in turn initialized by some random seeding process. While such approaches can yield (very) good optimizations of complex cost functions Q , their non-deterministic and dataset-specific behavior creates practical challenges, as described further in this section.

Parametric methods work *indirectly* over the projection. They fix some family of functions $P_\phi, \phi \in \Phi$, where ϕ is some set of parameters. Optimization then finds the best ϕ according to the method’s loss function Q . This single function P_ϕ is applied to every data point $\mathbf{x} \in \mathbf{X}$ to compute its projection $P(\mathbf{x})$. Differently from non-parametric methods, altering the position of one point typically affects other points as well, since the parameter vector ϕ itself is altered. Parametric methods have several key advantages. First and foremost, they can compute the projection of *new, unseen* data points — they effectively project the entire *space* $\mathbb{R}^n$ rather than a specific fixed *set* $\mathbf{X} \subset \mathbb{R}^n$. This allows one to construct P based on data $\mathbf{X}$ existing at some point in time and next augment the result with new points coming from the same distribution without having to recompute the entire projection (which would be slow but could also cause spurious changes in the resulting scatterplot). This property of parametric projections is also known in machine learning as *out of sample* ability — that is, the ability of a model to generalize its behavior outside a training set. Separately, a parametric method can be fit on a subset $\mathbf{X}_s \subset \mathbf{X}$ of the data, where $|\mathbf{X}_s| \ll |\mathbf{X}|$. The learned parameters can then be used to project the entire dataset $\mathbf{X}$. Using a representative subsample $\mathbf{X}_s$ yields typically small approximation errors. Projecting the entire dataset $\mathbf{X}$ can then be done using the learned parameters, providing major performance improvements.

Figure 2.2 illustrates the operation of a non-parametric projection — in this case, t-SNE (van der Maaten and Hinton, 2008) — compared to a parametric one — in this case, NNP (Espadoto et al., 2019a) (the latter is detailed further in Sec. 2.7.4). In both cases, we project a dataset having an increasing number of samples — from 2K to 100K — drawn from the same distribution, namely the well-known MNIST digit collection (LeCun et al., 1998). Colors annotate projected samples by the ten classes of MNIST. As visible, the non-parametric projection changes the location of the projected digit clusters wildly between the different projection scatterplots but, in return, achieves a good separation of these clusters (which is one of the goals of the underlying cost function Q considered by t-SNE). In contrast, the parametric projection keeps the location of these clusters well fixed

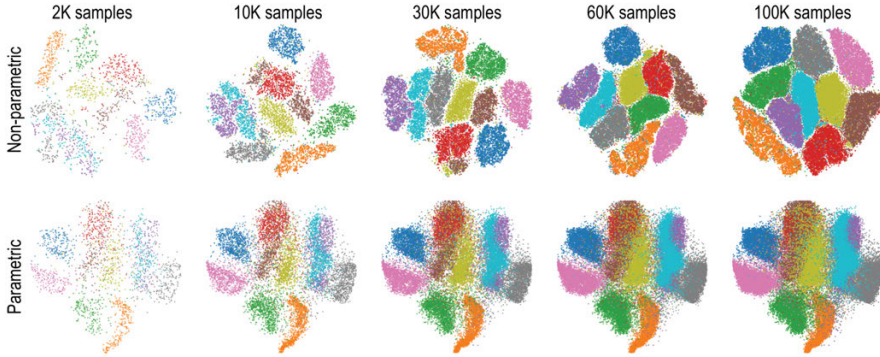


Figure 2.2: Comparative operation of non-parametric projections (t-SNE, top row) and parametric projections (NNP, bottom row) for a dataset having an increasing number of samples. Image adapted from (Espadoto et al., 2019a).

in the 2D projection plane as data samples are increasingly added to the input dataset; the price to pay for this stability is seen by the decreased separation, or fuzziness, of the different clusters in the projection. Further work has outlined that parametric projections are highly stable even upon significant alterations of their input data including adding significant noise and/or deletion of data dimensions (Bredius et al., 2022) or changing the projection’s hyperparameters (Appleby et al., 2022).

Several parametric projection methods exist. Early methods include PCA, NCA (Goldberger et al., 2004) and variants thereof. As outlined in Sec. 2.3.2, such methods compute linear transformations $\mathbf{X} \mapsto \mathbf{X}\mathbf{W}$, where $\mathbf{W}$ can also be applied to unseen data. However, their expressiveness is limited by their linear nature — that is, they cannot map well data patterns whose intrinsic dimensionality exceeds two. Other early parametric methods include self-supervised approaches such as Auto-Encoders (AEs, Hinton and Salakhutdinov, 2006). These can be used to learn projections by setting the number of neurons in the bottleneck of the network to q . The Encoder creates a projection to $\mathbb{R}^q$ and the Decoder outputs a reconstruction to $\mathbb{R}^n$. As plain AEs are typically poor at capturing local relationships and showing good class separation in projections, several extensions thereof have been proposed (Makhzani et al., 2015; Espadoto et al., 2021a; Machado et al., 2024e). These rely, among others, on pseudo-labels inferred from clustering or use Generative Adversarial Networks (GANs) to increase the desired class separation in the resulting projections. Still, such methods yield, in general, lower-quality projections than non-parametric approaches such as t-SNE or UMAP.

More recent parametric methods include Parametric t-SNE (p-TSNE, (van der Maaten, 2009)) which uses a multi-staged pipeline to build an ap-

proximation that preserves the local structure present in a computed t-SNE projection. This approach accurately extends t-SNE to a parametric setting with the main advantage of keeping the aforementioned high quality of the t-SNE projection. However, this method is computationally expensive (even more so than the already expensive underlying t-SNE technique); is complex to implement; and requires careful setting of many hyperparameters. Further parametric methods include supervised deep-learning approaches such as the NNP-class methods described further in Sec. 2.7.4.

Globally put, all parametric DR methods follow a *supervised* approach: Given some dataset $\mathbf{X} \in \mathbb{R}^n$, they construct a mapping $P : \mathbb{R}^n \rightarrow \mathbb{R}^q$ which aims to extrapolate the projection of $\mathbf{X}$ to the projection space $\mathbb{R}^q$. Once this mapping P is constructed, it is fixed, namely, it can be used for any point in $\mathbb{R}^n$ outside $\mathbf{X}$. Since P is fixed, the projection is deterministic; since P covers the entire $\mathbb{R}^n$ space, the projection is out-of-sample by construction; finally, since P is typically constructed so as to be smooth, the projection is stable, *i.e.*, exhibits small changes of the output scatterplot upon small changes of the input data.

The aforementioned trade-off between high projection quality but lack of stability achieved by non-parametric projections and, additionally, the high stability but lower projection quality achieved by parametric projections, have been conjectured to be a necessary price to pay for the ability of parametric projections to capture the entire data space (Espadoto et al., 2019a). Several studies have aimed at improving the quality of parametric projections — that is, decrease the observed fuzziness shown by the example in Fig. 2.2 — by exploring a wide range of parameter setting and different models for the function family P_ϕ to learn (Espadoto et al., 2020, 2021a; Modrakowski et al., 2022; Oliveira et al., 2023a), with limited results. Our work in Chapter 3 takes a step in this direction and shows that the advantages of parametric projections are not irreconcilable with decreasing the fuzziness observed by current parametric projection techniques.

2.4 PROJECTION QUALITY METRICS

Many projection techniques exist with different optimization aims and parameters, so one can easily create many different scatterplots $P(\mathbf{X})$ from the same data $\mathbf{X}$ (Wattenberg et al., 2016; Vu et al., 2021; Cutura et al., 2018). More formally: As mentioned earlier in Sec. 2.3, each projection technique P constructs the scatterplot $P(\mathbf{X})$ by optimizing, either explicitly or implicitly, for a cost function Q . In turn, different expressions of this cost Q imply that, given the same dataset $\mathbf{X}$, one can create different projection scatterplots $P(\mathbf{X})$. In turn, these different scatterplots will exhibit different so-called *visual patterns* that map, up to different degrees, various aspects, or *data patterns*, present in the input dataset $\mathbf{X}$. There is — both theoretically and practically — an infinite set of possibilities for a projection scatterplot

$P(\mathbf{X})$ to capture, or map, certain data patterns from $\mathbf{X}$ to the visual space. However, it is equally clear that not all such mappings are equally good, regardless of the task one will use them for. As such, it is clear that having an *objective* way to quantify what, and how well, a scatterplot $P(\mathbf{X})$ maps from a given $\mathbf{X}$ is highly desired.

Projection Quality Metrics (PQMs) are the currently accepted way to perform the above quantification. PQMs are functions that measure how well certain data patterns in $\mathbf{X}$ are mapped to some visual patterns in $\mathbf{Y} = P(\mathbf{X})$. In short, they attribute a scalar score to a projection $\mathbf{Y}$, given original data $\mathbf{X}$ and possibly label information C . Formally put, a quality metric is a function

$$\begin{aligned} \mathcal{M}_P : \mathcal{X} \times \mathcal{P}^q &\rightarrow \mathbb{R}^+ \\ (\mathbf{X}, P(\mathbf{X})) &\mapsto x \end{aligned} \tag{2.1}$$

that maps a dataset $\mathbf{X}$ and its projection scatterplot $P(\mathbf{X})$ computed by a given technique P to a real value x .

Several such metrics exist, measuring the amount of preservation of different data patterns in the projection. A useful PQM should, as much as possible, identify good projections of a given dataset $\mathbf{X}$ and rule out poor ones. For example, the Stress metric (Kruskal, 1964) measures the amount of distortion in pairwise distances introduced by P . High values mean P distorts the high-dimensional distances, possibly favoring the preservation of neighborhood structures — measured by other metrics such as Trustworthiness and Continuity (Venna and Kaski, 2006).

2.4.1 Types of projection quality metrics

In this thesis, we classify PQMs into *global* and *local* ones, as follows.

Global Projection Quality Metrics are those that output a *single* scalar — *i.e.*, a single ‘judgment’ — for the projection. That is to say, the scope of these PQMs is the *whole* projection at once. Global metrics are also useful for ranking projection techniques in automatic benchmarks — one can, for example, compute such metrics for several projection techniques on a (large) collection of datasets and next determine which techniques score highest. Similar procedures are used routinely when ranking machine learning (ML) algorithms and involved global ML metrics such as accuracy, Cohen’s kappa score, or area under the ROC curve (Hand, 2009). While global metrics are simple to compute and interpret, they nevertheless suffer from the limitations of descriptive statistics mentioned in Sec. 1.1 — for example, they cannot show whether a projection technique captures well certain data patterns for a part of the input dataset but utterly fails in doing so for another part of the data.

Local PQMs, on the other hand, are able to emit a quality judgment for the mapping of *each projection point* $\mathbf{y}_i = P(\mathbf{x}_i)$ or, more generally, for small neighborhoods defined around these points. Every local PQM admits, evidently, a global version — created by, for example, averaging the quality values of each point. Local PQMs can be used to train regression algorithms, which require one target value *per data point*, something that is incompatible with global PQMs. We exploit this property in Chapter 5. Separately, local PQMs can be used to generate visualizations of projection quality which allow users to inspect where, and how much, a given projection captures certain data patterns (Martins et al., 2014). In turn, such visualizations can be effectively used to *e.g.* interpret a projection scatterplot in a more nuanced all-or-nothing fashion — for instance, one can dismiss poorly projected points from such a scatterplot and focus one’s analysis on the remaining points (Rodrigues et al., 2019).

Several other classifications of PQMs exist in the literature. For instance, Espadoto et al. (2021b) identify, apart from global and local metrics, so-called point-pair metrics, which characterize pairs of points in the projection scatterplot. Other authors define metrics for clusters of points (Sedlmair et al., 2012; Tatu et al., 2010). For our goals in this thesis, the simpler characterization in global and local metrics suffices, so we do not consider such further refinements. Separately, we note that several libraries have also been proposed to compute such metrics, with an eye on favoring quality-assessment replicability and benchmarking DR algorithms (Jeon et al., 2023; Espadoto et al., 2021b). Our own work in Chapter 5 contributes to this direction by proposing a reusable library of 17 such metrics (Machado et al., 2025a).

Finally, we want to draw a distinction between PQMs and Visual Quality Metrics (VQMs, (Bertini and Santucci, 2006)). The latter can be used to assess the purely-visual patterns of a scatterplot which may represent the projection of some dataset but, also, can be created in many other different ways. As such, VQMs are not directly adequate to judge a projection $P(\mathbf{X})$ in terms of how well it preserves patterns *from the data* $\mathbf{X}$. For instance, a VQM may characterize some scatterplot $P(\mathbf{X})$ as being of high visual quality — due to the presence of certain patterns — even though such patterns do not map in any measurable way data patterns in $\mathbf{X}$. We further elaborate on this topic in our work in Chapter 5.

2.4.2 Metric definitions

We further detail the concrete projection quality metrics which we use in our work in this thesis. We have selected these from the wider set of metrics known in the literature following largely the rationale outlined by Espadoto et al. (2021b). That is, we focus on metrics which are simple to compute, can be computed automatically (without user intervention),

generically handle any dataset and projection technique, and have a limited number of simple-to-set hyperparameters. This excludes, for instance, metrics which depend on complex clustering strategies or metrics which involve user experiments (Abbas et al., 2021, 2019).

To define these metrics, a few preliminary notations and definitions are needed, as follows. We represent the label of data point $\mathbf{x}_i$ by c_i . $\mathbf{S}_D^k(\mathbf{x})$ is the set of k -nearest neighbors of a data point $\mathbf{x}$ in D -dimensional space (the data space); $\mathbf{S}_2^k(\mathbf{y})$ is the set of k -nearest neighbors of the projection $\mathbf{y}$ of data point $\mathbf{x}$ in the projected space (*i.e.*, in the generated scatterplot). We abbreviate $\delta_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$ and $\bar{d}_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|$. For some metrics, pairwise distances are first normalized to be between 0 and 1, which we represent by a bar on top of the respective symbol, *e.g.*, $\bar{\delta}_{ij} = \frac{\delta_{ij} - \min_{i,j} \delta_{ij}}{\max_{i,j} \delta_{ij} - \min_{i,j} \delta_{ij}}$. We denote by $r(i, j)$ the rank of $\mathbf{x}_j$ in the list of points sorted according to distances to $\mathbf{x}_i$ in the data space. Analogously, we denote by $\hat{r}(i, j)$ the rank of $\mathbf{x}_j$ in the list of points sorted according to distances to $\mathbf{x}_i$ in projection space. We also denote by $\llbracket \cdot \rrbracket$ the indicator function, which returns 1 if the condition inside it is true, and 0 otherwise. Finally, note that all these metrics can be computed both globally (as in the definitions listed below) but, also, locally, by restricting the quantities under their respective summations to desired point subsets.

With these notations, our considered metrics are as follows.

Average Local Error: Measures the average absolute pairwise distance distortion generated by P (Martins et al., 2014).

$$\mathcal{M}_{\text{ALE}} = \frac{1}{m(m-1)} \sum_i \sum_{j \neq i} |\bar{\delta}_{ij} - \bar{d}_{ij}|. \quad (2.2)$$

Continuity: Penalizes missing neighbors of each projected point. Missing neighbors are those elements that *are* in the k nearest neighbors (k NN) set of a point in the data space but *are not* among its k NN in the projection (Venna and Kaski, 2006; Martins et al., 2014)

$$\mathcal{M}_{\text{Cont}} = 1 - \frac{1}{m} \sum_i \frac{2}{k(2m - 3k - 1)} \sum_{j \in \mathbf{S}_D^k - \mathbf{S}_2^k} (\hat{r}(i, j) - k). \quad (2.3)$$

Stretching and compression (Aupetit, 2007; Lespinats and Aupetit, 2011) are two metrics which capture very similar aspects to continuity, respectively trustworthiness (described further below).

Class-Aware Continuity: Penalizes missing neighbors of each projected point $\mathbf{y}$ *only if* the missing neighbor is of the same class as $\mathbf{y}$ (Colange et al., 2020).

$$\mathcal{M}_{\text{CACont}} = 1 - \frac{1}{m} \sum_i \frac{2}{k(2m - 3k - 1)} \sum_{j \in \mathbf{S}_D^k - \mathbf{S}_2^k} \llbracket c_j = c_i \rrbracket (\hat{r}(i, j) - k). \quad (2.4)$$

Distance Consistency: One level higher than k -nearest neighborhoods, one can quantify projections of labeled data at the *class* level. Visual separation metrics (Albuquerque et al., 2011; Motta et al., 2015; Sedlmair et al., 2013; Benato et al., 2023) gauge how well points from each class, in a projection, are visually separated from points of other classes. The underlying idea is that ground-truth information on the class separation in $\mathbf{X}$ can be used to compute how well $P(\mathbf{X})$ reflects that separation. Sedlmair and Aupetit (2015) surveyed 15 such metrics. Their conclusion is that the distance consistency metric (DSC, (Sips et al., 2009)), also presented as class consistency measure (CCM, (Tatu et al., 2010)), best encodes the perception of class separation in labeled scatterplots. DSC is computed assuming that, in a well-separated labeled dataset D , a point with label c is closest to the centroid (barycenter) of all points with the same label c (which we denote as $K(c)$) than to centroids of other classes. A good projection should keep this property on $P(\mathbf{X})$. DSC is computed as

$$\mathcal{M}_{\text{DSC}} = \frac{1}{m} \sum_i \mathbb{I}[c_i = \arg \min_{c'} \|\mathbf{y}_i - K(c')\|]. \quad (2.5)$$

Jaccard: Measures the accuracy of projection neighborhoods as intersection-over-union of the sets of projection neighbors and data space neighbors (Jaccard, 1901)

$$\mathcal{M}_{\text{J}} = \frac{1}{m} \sum_i \frac{|\mathbf{S}_D^k \cap \mathbf{S}_2^k|}{|\mathbf{S}_D^k \cup \mathbf{S}_2^k|}. \quad (2.6)$$

Mean Relative Rank Errors Data/Projection: Measures the relative rank errors in the k NN sets of each projected point. The ‘Data’ version uses the data space as reference, whereas the ‘Projection’ version uses the projection itself (Lee and Verleysen, 2009).

$$\mathcal{M}_{\text{MRREData}} = \frac{1}{m} \sum_i \sum_{j \in \mathbf{S}_2^k} \frac{|r(i, j) - \hat{r}(i, j)|}{\hat{r}(i, j)} \quad (2.7)$$

$$\mathcal{M}_{\text{MRREProj}} = \frac{1}{m} \sum_i \sum_{j \in \mathbf{S}_D^k} \frac{|r(i, j) - \hat{r}(i, j)|}{r(i, j)}. \quad (2.8)$$

Neighborhood Hit: Measures the proportion of points in the k NN of each projected point $\mathbf{y}$ that have the same class as $\mathbf{y}$ (Paulovich et al., 2008):

$$\mathcal{M}_{\text{NH}} = \frac{1}{m} \sum_i \frac{1}{k} \sum_{j \in \mathbf{S}_2^k} \mathbb{I}[c_j = c_i]. \quad (2.9)$$

Normalized Stress: Measures the squared distortions in pairwise distances introduced by P (Kruskal, 1964; Smelser et al., 2024). This metric is ‘normalized’ by the distances in the high-dimensional space. The name ‘normal-

ized' is a misnomer: this metric is unbounded and its best value is zero, corresponding to no distortion.

$$\mathcal{M}_S = \frac{\sum_{i,j} (\delta_{ij} - d_{ij})^2}{\sum_{i,j} \delta_{ij}^2}. \quad (2.10)$$

The projection precision score (Schreck et al., 2010), which measures how the distances from a data point to its k -nearest neighbors differ from the analogous distances between their scatterplot points, is a neighborhood-level adaptation of $\mathcal{M}_S$.

Pearson Correlation: The Pearson correlation of the distances δ_{ij} and d_{ij} .

$$\mathcal{M}_{\text{PCorr}} = \text{Correlation}(\delta_{ij}, d_{ij}). \quad (2.11)$$

The interpretation of this metric is simple: The higher the distances d_{ij} in the projection space are correlated to distances δ_{ij} in the data space, the easier can an analyst visually explore the former to infer the latter. That is, $\mathcal{M}_{\text{PCorr}}$ is a way to measure the distance preservation of a projection in an aggregated way (Geng et al., 2005).

Procrustes Statistic: The average of the computed Procrustes statistic across the entire projection (Goldberg and Ritov, 2009). The Procrustes statistic computes per-neighborhood best matchings of point clouds by rotating and scaling them to match each other. Denoting by $\mathbf{H}$ an appropriately-sized centering matrix — *i.e.*, $\mathbf{H} = \frac{1}{n}\mathbf{1}\mathbf{1}^T$ — the metric is given by

$$\mathcal{M}_{\text{Proc}} = \frac{1}{m} \sum_i \frac{G(\mathbf{S}_D^k(\mathbf{x}_i), \mathbf{S}_2^k(\mathbf{y}_i))}{\|\mathbf{H} \mathbf{S}_D^k(\mathbf{x}_i)\|_{\text{Fro}}^2}, \quad (2.12)$$

where $G(\mathbf{X}, \mathbf{Y}) = \|\mathbf{H}(\mathbf{X} - \mathbf{Y}\mathbf{A}^T)\|_{\text{Fro}}^2$, $\mathbf{A} = \mathbf{U}\mathbf{V}^T$, and $\mathbf{U}\mathbf{L}\mathbf{V}^T = \text{SVD}(\mathbf{X}^T\mathbf{H}\mathbf{Y})$.

Scale-Normalized Stress: A scale-insensitive version of the Normalized Stress metric (Smelser et al., 2024). The computation is the same as for Normalized Stress, except the projection space distances d_{ij} are scaled by a factor α :

$$\mathcal{M}_{\text{SNS}} = \frac{\sum_{i,j} (\delta_{ij} - \alpha d_{ij})^2}{\sum_{i,j} \delta_{ij}^2}, \text{ where } \alpha = \frac{\sum_{i,j} \delta_{ij} d_{ij}}{\sum_{i,j} d_{ij}^2}. \quad (2.13)$$

Shepard Goodness: The Spearman rank correlation of the distances δ_{ij} and d_{ij} (Siegel and Castellan, 1988; Joia et al., 2011)

$$\mathcal{M}_{\text{SGood}} = \text{RankCorrelation}(\delta_{ij}, d_{ij}). \quad (2.14)$$

True/False Neighbors Rate: Measures the proportion of points in the k NN of each projected point that correspond (resp. do not correspond) to neighbors in the data space (Martins et al., 2014)

$$\mathcal{M}_{\text{TN}} = \frac{1}{m} \sum_i \frac{|\mathbf{S}_D^k \cap \mathbf{S}_2^k|}{k} \quad (2.15)$$

$$\mathcal{M}_{\text{FN}} = 1 - \mathcal{M}_{\text{TN}}. \quad (2.16)$$

Trustworthiness: Penalizes false neighbors of each projected point. False neighbors are those elements that *are not* in the k NN set of a point in the data space but *are* among its k NN in the projection (Venna and Kaski, 2006; Martins et al., 2014). This metric is the ‘converse’ of Continuity.

$$\mathcal{M}_{\text{Trust}} = 1 - \frac{1}{m} \sum_i \frac{2}{k(2m - 3k - 1)} \sum_{j \in \mathbf{S}_2^k - \mathbf{S}_D^k} (r(i, j) - k). \quad (2.17)$$

Class-Aware Trustworthiness: Penalizes false neighbors of each projected point $\mathbf{y}$ *only if* the false neighbor is of a different class than $\mathbf{y}$ (Colange et al., 2020).

$$\mathcal{M}_{\text{CATrust}} = 1 - \frac{1}{m} \sum_i \frac{2}{k(2m - 3k - 1)} \sum_{j \in \mathbf{S}_2^k - \mathbf{S}_D^k} \mathbb{I}[c_j \neq c_i] (r(i, j) - k). \quad (2.18)$$

2.5 INTERPRETING VISUAL PATTERNS IN PROJECTIONS

As outlined in Sec. 2.4, different projection techniques map a dataset $\mathbf{X}$ to a scatterplot $P(\mathbf{X})$, yielding different visual patterns. Users that see such visual patterns can then ‘reverse’ the mapping to infer the associated data patterns (Munzner, 2014; Telea, 2014). Such patterns include (1) clusters of densely packed points with similar values for a dimension, shown, *e.g.*, via color coding; (2) clusters with different point densities; (3) overlaps of clusters with different point colors; (4) clusters of different elongations or shapes; or (5) outlier points far from the scatterplot bulk. Such visual patterns have been studied in detail by *scagnostics* works for selecting informative scatterplots (Tukey and Tukey, 1988; Wilkinson et al., 2005, 2006; Yates et al., 2014). Visual patterns help inferring data insights, or properties, such as (1) similar samples sharing a common (dependent) attribute, *e.g.*, label; (2) data subsets of different internal similarities; (3) mixed distribution modes yielding different values for a dependent attribute; (4) latent variables along which data has different variance; and (5) samples not obeying a standard distribution (Behrisch et al., 2018; Pandey et al., 2016; Wang et al., 2018).

Following this, it means that a projection technique that creates visual patterns which can be easily (and correctly) interpreted is of high quality.

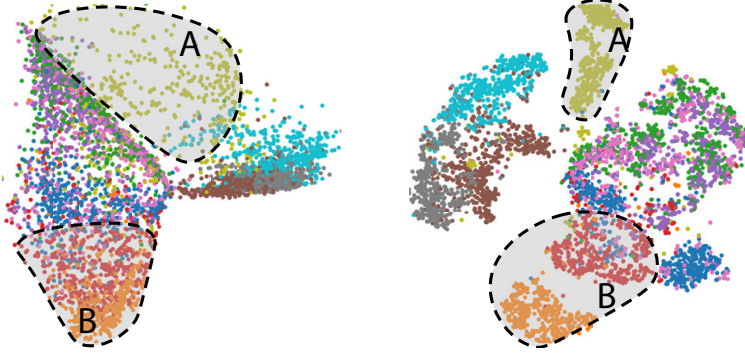


Figure 2.3: Isomap (left) and t-SNE (right) projections of the FashionMNIST dataset (Xiao et al., 2017) showing contradictory visual patterns. In (A), Isomap shows same-class points spread out, suggesting they are not very similar; t-SNE packs these points tightly, suggesting the opposite. In (B), Isomap shows the red and orange classes overlapping; t-SNE shows these classes clearly separated.

In parallel, we have seen that projection quality metrics aim to measure the quality of a projection (Sec. 2.4). One may hereby conclude that PQMs can be used as a tool to measure the ability of a projection to depict visual patterns which are useful and usable for data analysis. Yet, this is not fully the case for several reasons. First, we note that most, if not all, projection quality metrics operate on an either *very local* level or on a *fully global* one. Visual patterns, however, exist on a multitude of scales between such levels, and hence are hard to capture by existing PQMs. Secondly, we see that current PQMs measure quite simple properties of the projection scatterplot, *e.g.*, distance preservation, similarity of labels of close points, or neighborhood preservations. Visual patterns span a far richer space than what such simple metrics can capture. In other words: PQMs are *necessary* quality indicators that tell us if a projection can be interpreted to extract data insights; however, they do not tell us which *exact* data insights the visual patterns in a projection do capture.

A subsequent question we ask ourselves is how do different projection *techniques* preserve specific data patterns. If different techniques show wildly different patterns when operating on the same dataset $\mathbf{X}$, it stands to reason that some projections will be better able to preserve certain types data patterns. Figure 2.3 confirms this hypothesis by a simple experiment: We see here the same dataset, FashionMNIST, projected by the Isomap and t-SNE techniques. The two projections show actually contradictory visual patterns.

Earlier works in DR literature have also noted this phenomenon. More specifically, certain projection techniques are known to produce specific

visual patterns *independent* of the dataset being projected. For example, t-SNE tends to produce clusters that are spread out roughly filling a circle in the projection, and that these clusters are separated by a uniform amount of white space. UMAP tends to produce clusters that are tightly packed, widely separated with white space. Auto-Encoder based projections tend to create starburst-like with streaks emanating from the center of the projection. Isomap projections tend to exhibit line-like structures (Tenenbaum et al., 2000). This consistent way of laying out points in the projection *across datasets* is what we call the *visual signature* of a projection technique. Each DR algorithm’s signature stems from its mathematical underpinnings, as well as its implementation details and hyperparameter settings.

While such visual signatures clearly influence what users perceive and interpret in a projection scatterplot, only limited work was done into controlling and/or influencing the visual signature of projections with goals such as readability or preservation of finer/coarser structures. Cutura et al. (2022) use space-filling curves to adapt the position of data points in image thumbnail scatterplots such that they are non-overlapping. This can be seen as a local way to control the generation of visual patterns in a projection. The perplexity parameter in t-SNE controls, among other things, a projection’s visual appearance. Its effect is intricately enmeshed within t-SNE, leading to cluster shapes, sizes, and distances that do not necessarily convey meaning (Wattenberg et al., 2016). Related to this, since fully doing away with hyperparameters might be infeasible, techniques such as HyperNP (Appleby et al., 2022) learn to simulate their effects on the resulting projection. This helps users quickly scan different perplexity values to choose the one generating a projection with the desired visual signature. Yet, this does not let users directly *impose* that a projection uses a specific visual signature. Finally, Makhzani et al. (2015) adapt an auto-encoder into an adversarial setting to increase the quality of the projection (*e.g.*, better cluster separability) that a vanilla auto-encoder would create. We further detail work on exploring visual signatures of projections in Sec. 4.1.2.

Our work contributes to the understanding of projection visual signatures in two separate ways. First, we present in Chapter 4 a projection technique that explicitly allows users to control the *shapes* of clusters generated in the projection scatterplot, independently of the input data. This eliminates one important confusion factor that current projection techniques have — that is, users of our technique know that the shapes of the visual patterns they see in the projection do not encode actual data patterns (and can do away with the variability of such shapes by explicitly controlling them). Secondly, we show in Chapter 5 that existing projection quality metrics are necessary, but not *sufficient* predictors for the quality of a projection, *i.e.*, its ability to effectively capture data patterns.

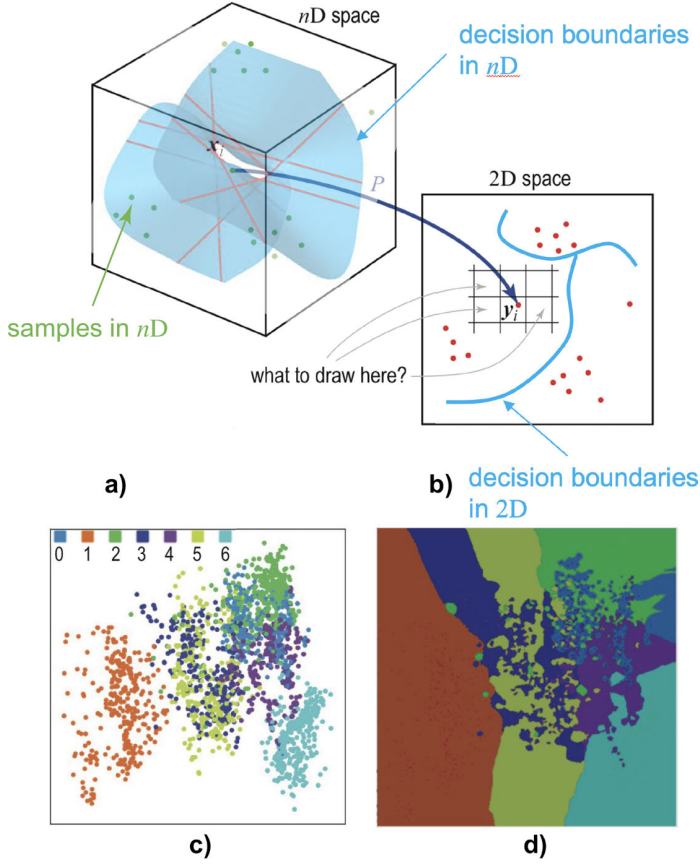


Figure 2.4: Principle of computing and visualizing decision boundary maps (DBMs) for visualizing the behavior of trained classification models. Decision zones and boundaries in data space (a) can be mapped to projection space (b). Classical scatterplot visualization of a classifier (c) can be augmented to show decision zones and boundaries (d). Image taken from (Telea et al., 2024).

2.6 INVERSE PROJECTION AND DECISION BOUNDARY MAPS

The ability to construct a so-called ‘direct’ projection mapping P from $\mathbb{R}^n$ to $\mathbb{R}^q$ raises the question whether constructing an inverse mapping

$$P^{-1} : \mathbb{R}^q \rightarrow \mathbb{R}^n \quad (2.19)$$

is possible and, if so, useful. Recent work has shown that such inverse mappings can be constructed by minimizing the cost $\|P^{-1}(P(\mathbf{x})) - \mathbf{x}\|$, $\mathbf{x} \in \mathbf{X}$ for a given dataset $\mathbf{X}$ by using methods such as feedforward neural networks (NNInv, Espadoto et al., 2019b) or autoencoder-based strategies —

plain ones (Hinton and Salakhutdinov, 2006) and, for the more recent SSNP method, using labels provided by the dataset or computed via clustering (Espadoto et al., 2021a). Additional techniques handle inverse projection by focusing on inverting the effect of specific direct projections — for example, iLAMP (dos Santos Amorim et al., 2012) uses radial basis functions to invert the effect of the LAMP projection (Joia et al., 2011); MultiInv handles the inversion of MDS (Blumberg et al., 2025); while the UMAP technique provides both direct and inverse projection in the same framework (McInnes and Healy, 2018).

It is important to note that an inverse projection P^{-1} is not the mathematical inverse function of the projection function P for several reasons: First and foremost, as explained earlier in Sec. 2.3, in the non-parametric case, P may not be defined as a *function* from $\mathbb{R}^n$ to $\mathbb{R}^q$ but rather defined only for a given dataset $\mathbf{X} \in \mathbb{R}^n$; secondly, the mapping P (whether in parametric or non-parametric case) may not be injective (Wang et al., 2023).

Inverse projections have many practical uses such as shape and image morphing (dos Santos Amorim et al., 2012) and data imputation (Espadoto et al., 2023a). In short, they allow us to *sidestep* the sparsity of the dataset $\mathbf{X}$ used during direct projection. In particular, consider any application in which $\mathbf{X}$ is visualized by a projection scatterplot. Typically, such a dataset $\mathbf{X}$ only captures a *sampling* of some dense phenomenon taking place in $\mathbb{R}^n$. As such, when visualizing the scatterplot $P(\mathbf{X})$, analysts may wonder what the said phenomenon behaves like when sampled at locations that correspond, via projection, to points falling in the gaps, or white space, *between* or *around* locations in $P(\mathbf{X})$. With an inverse projection, we can assign a synthetic data point $P^{-1}(\mathbf{y})$ for all $\mathbf{y} \in \mathbb{R}^q$, evaluate our phenomenon at location $P^{-1}(\mathbf{y}) \in \mathbb{R}^n$, and depict the result at $\mathbf{y}$. This yields a dense visualization of our phenomenon that covers all locations (pixels in 2D) in the projection space.

Such dense visualizations come in as an important tool for the study of trained classification models. Indeed, consider the typical evaluation of such models: Numeric aggregate metrics — such as accuracy, F-measure, area under ROC curve — are valid approaches to evaluate a classifier but provide little in the sense of *exploring* how a classifier’s behavior changes across different regions of the data space. Such metrics typically allow an investigator to have either a global view — each metric outputs a single score for the classifier as a whole — or an unaggregated, per data point, view. This is an identical limitation to the one exposed for projection quality metrics (PQMs) in Sec. 2.4.1 — this time, however, it affects the study of classification models. This introduces a disconnect in the pipeline: classifiers are *total* functions, able to output classifications for any point in their training data space. However, data points only *sparsely* sample this space, making any approach that relies solely on them limited by design on the

type of insights it can produce: it cannot say what happens in the (quite large) unsampled portions of the data space.

One way to address this limitation is to evaluate the classifier over dense and regular samplings of the data space. *Decision boundary maps*, or DBMs, do precisely that (see Fig. 2.4). Given a trained classification model $f : \mathbb{R}^n \rightarrow K$, classical ways to explore the operation of the model involve selecting a dataset $\mathbf{X} \in \mathbb{R}^n$ and visualizing the projection $P(\mathbf{X})$ color coded by the decisions of f (see Fig. 2.4c). However, this does not show the exact so-called *decision zones* of the model, *i.e.*, the areas in data space where f predicts the same class label, nor the so-called *decision boundaries*, *i.e.*, the hypersurfaces in data space where f changes value (see Fig. 2.4a). Decision maps, while not able to capture the entire complexity of decision zones and boundaries, offer nevertheless a simplified, easy-to-interpret visual approximation to such structures (Fig. 2.4b). Decision zones appear, in the projection space, as same-color areas; decision boundaries are indicated by pixels with differently colored neighbors (Fig. 2.4d).

Several methods have been proposed for the computation of DBMs (Rodrigues et al., 2018; Oliveira et al., 2022; Schulz et al., 2015, 2020; Oliveira et al., 2022; Blumberg et al., 2025). Acceleration techniques have been also proposed to reduce the number of evaluations of the inverse projection P^{-1} needed to compute high-resolution DBMs (Wang et al., 2025). A recent study quantitatively compared such methods along several quality metrics designed with similar goals as the quality metrics used for direct projections discussed in Sec. 2.4 (Wang et al., 2023).

Separately, the limitations of decision map methods with respect to their coverage of the data space have been analyzed (Wang and Telea, 2024). A key finding of this study is that decision map techniques in existence only cover a *surface* embedded in the data space of the studied classifier. From a practical perspective, decision map visualizations have been demonstrated to effectively help the improvement of classification models by assisting users to efficiently annotate unlabeled datasets (Benato et al., 2024b). Separately, *gradient maps*, computed using a similar principle to decision maps to visualize the norm of the gradient of P^{-1} (Espadoto et al., 2023a), have been proposed to identify regions in projected space where inverse projections, or derived quantities such as DBMs, may be inaccurate and thus unreliable. We refine this technique in our work in Chapter 6 to propose additional maps that enhance the interpretation of plain DBMs. Finally, decision maps have also been extended to visualize the output of real-valued regressors (Espadoto et al., 2023b).

While decision maps bring additional insights into the behavior of classification models over raw, sparse, class-label colored scatterplots, the fact that they only capture a subset of the data space (as shown by Wang and Telea, 2024) and that inverse projections are highly non-linear (as shown by Espadoto et al., 2023a) can create misleading insights. We explore this as-

pect in our work in Chapter 6 and show that plain DBMs can, indeed, lead to false interpretations of how classification models operate. Along this, we propose enhancements of plain DBMs that alleviate this problem by depicting additional information extracted from the studied models. This is our key contribution to the topic ‘seeing for learning’ mentioned in the beginning of this chapter.

Finally, let us note that DBMs are just one example of a far wider set of visual analytics (VA) techniques that fall within the scope of what is known under the name Explainable AI (XAI). In general, XAI techniques address far broader aspects than depicting the operation of a classification model; these include the analysis of how specific data dimensions and/or data values influence a model’s prediction; the analysis of how specific elements of neural architectures (weights, layers, neural units, training procedure elements) affect a model’s performance; the explanation of complex models via simpler surrogates; and the integration of additional aspects such as trust in exploring ML models. It is out of this thesis’ scope, which focuses exclusively in this respect on enhancing DBMs, to cover all such aspects. We point the interested readers in XAI for VA to recent surveys in this area (Alicioglu and Sun, 2022; Choo and Lu, 2018; Rosa et al., 2023; van den Elzen et al., 2023).

2.7 MACHINE LEARNING

In this section, we provide a brief summary of the key concepts related to machine learning which are necessary to further delineate the contributions of our work. Specifically, we introduce the basis of classification (Sec. 2.7.1) and regression (Sec. 2.7.2) models; outline the process of regularization (Sec. 2.7.3); and outline how machine learning, and in particular deep learning, has recently contributed to improve (inverse) dimensionality reduction techniques (Sec. 2.7.4).

2.7.1 Classification

A *classifier* is a type of Machine Learning (ML) model trained to predict, as accurately as possible and in generalizable fashion, the *class* (also called *label* or *category*) of a given input. In this work, we denote classifiers by $f_{\theta} : \mathbb{R}^n \rightarrow [K]$, where the *label set* is represented by $[K] = \{1, 2, \dots, K\}$, the set of naturals not greater than K . Each number in $[K]$ corresponds to a specific class f_{θ} aims to identify — for example, 1 for Dog and 0 for Cat —, and the class of a specific data point $\mathbf{x}$ can be queried through the function $C(\mathbf{x})$. We use this notation to avoid confusion with the usual representation for low-dimensional projections $\mathbf{y} = P(\mathbf{x})$. Therefore, in this notation, the training dataset for a classifier can be written $\mathcal{D} = \{(\mathbf{x}_i, C(\mathbf{x}_i))\}_{i=1, \dots, n}$ or

equivalently $\mathcal{D} = \{(\mathbf{x}_i, c_i)\}_{i=1,\dots,n}$. We represent the classifier's parameters — when it is indeed parametric, such as an Artificial Neural Network — by θ ; in this case, θ is the set of weights and biases of each layer. Whenever clear from context, we omit θ for brevity and write simply f for a given classifier.

Classifiers can range from simpler ones, such as Logistic Regression — where the decision is based on linear combinations of input features which might themselves be non-linear — to more complex machines such as Deep Neural Networks. DNNs — or NNs for short — work by learning sequential linear transformations of data points interspersed with non-linear function applications. While each transformation is simple, the stacking of many transformations allows for state of the art performance in many classification tasks.

$$\mathbf{a}_0 = \mathbf{x} \quad (2.20)$$

$$\mathbf{h}_l = \mathbf{W}^{(l)} \mathbf{a}_{l-1} + \mathbf{b}^{(l)}, \quad l \in \{1, \dots, L\} \quad (2.21)$$

$$\mathbf{a}_l = \sigma(\mathbf{h}_l), \quad l \in \{1, \dots, L-1\} \quad (2.22)$$

$$\mathbf{a}_L = \mathcal{S}(\mathbf{h}_L), \quad (2.23)$$

where σ is a non-linear activation function such as the Rectified Linear Unit (ReLU) or the Logistic Sigmoid function and $\mathcal{S}$ is the vector-valued Softmax function

$$\mathcal{S}_i(h_1, h_2, \dots, h_k) = \frac{\exp(h_i)}{\sum_{j=1}^k \exp(h_j)}. \quad (2.24)$$

The weights and biases constitute the parameters of such models, $\theta = \{\mathbf{W}^{(l)} \mid l \in [L]\} \cup \{\mathbf{b}^{(l)} \mid l \in [L]\}$.

In the general multiclass case, NNs are trained to maximize classification accuracy by minimizing the Cross Entropy Loss

$$\mathcal{L}_{\text{CE}} = - \sum_i \sum_{j \in [K]} \mathbb{I}[c_i = j] \log f_j(\mathbf{x}_i). \quad (2.25)$$

The classifier-attributed probability that $\mathbf{x}$ has class j is represented by $f_j(\mathbf{x})$. Training is typically carried out with gradient-based optimizers such as Stochastic Gradient Descent (SGD) or Adam (Kingma and Ba, 2015).

2.7.2 Regression

Regressors are other kinds of Machine Learning models. These are trained to predict, in generalizable manner, one or multiple *scalar* signals — as opposed to categories in the case of classifiers. For example, we might be interested in predicting the price of a house given its characteristics. We use the same notation for regressors as for classifiers.

Regression models are of special interest to this work since Dimensionality Reduction algorithms themselves output one scalar per projection dimension for each of their inputs. In other words, the output space of a projection function P is $\mathbb{R}^q$, a real vector space. That allows us to cast the problem of *imitating* or *approximating* projections as one of regression.

Unsurprisingly, the use of Neural Networks for regression problems is also well-posed. They possess the same structure as classification networks, except for their last layer, where an identity function is used as ‘activation’ function ($\mathbf{a}_L = \mathbf{h}_L$). Such networks are typically trained with a Mean Squared Error loss

$$\mathcal{L}_{\text{MSE}} = \sum_i (\mathbf{y}_i - f(\mathbf{x}_i))^2. \quad (2.26)$$

Alternatively, a Mean Absolute Error loss can be used

$$\mathcal{L}_{\text{MAE}} = \sum_i |\mathbf{y}_i - f(\mathbf{x}_i)|. \quad (2.27)$$

2.7.3 Regularization

The more powerful a model is, the better it is at capturing even the finest patterns in the data. While that is a desirable property, it also leads to issues where the model can fit the true data-generating patterns *as well as* the noise contained in the observations making up the dataset. For example, a single outlier can throw off a simple linear regression model if it is given just as much importance as the in-distribution samples.

This leads up to an increase in the *generalization error* of machine learning models. The model is capable of (often perfectly) fitting the training dataset, but becomes so attuned to it that it ends up making mistakes on unseen test data. This is the problem of *overfitting*.

A remedy for overfitting is to use some sort of *regularization*, typically in the form of a complexity penalty. For Neural Networks, regularization is typically done by penalizing the L2 and/or L1-norm of the weights and biases of the network. The L2 norm penalty promotes finding weight matrices and bias vectors with smaller norms overall, leading to simpler models. The L1 norm induces sparsity, setting small magnitude elements in the weight matrices and bias vectors to zero.

2.7.4 Deep learning for dimensionality reduction

We have already detailed how dimensionality reduction techniques can assist machine learning tasks — in particular by the visualization of high-dimensional datasets via projections and the exploration of trained classification models via decision maps (Sec. 2.6). The converse is also true —

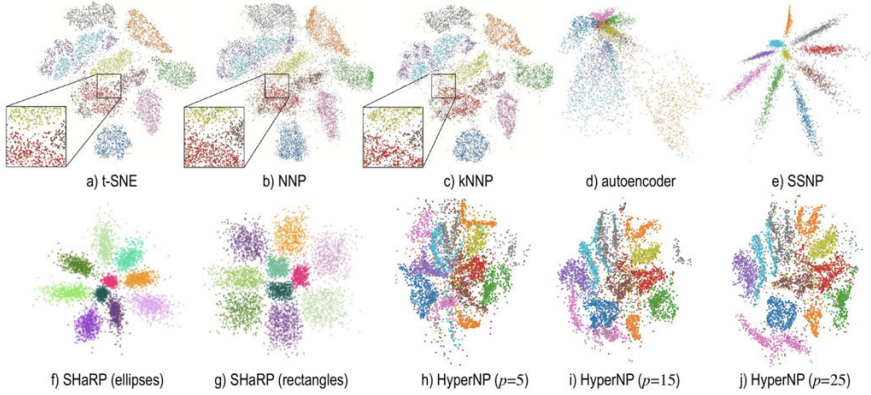


Figure 2.5: Deep learning for dimensionality reduction. A dataset is projected by a classical non-parametric technique (t-SNE, a). The same dataset is projected via various neural dimensionality reduction techniques (b-j) yielding a variety of visual patterns and projection styles. Image taken from (Telea et al., 2024).

that is, machine learning approaches, and in particular deep learning regressors, can be effectively used to create (inverse) projections. We outline below the main developments in this area.

Early on, Hinton and Salakhutdinov (2006) have proposed using autoencoders to perform dimensionality reduction of a high-dimensional dataset in a self-supervised way. This approach already showcased some of the key advantages of deep learning for DR, namely simplicity of implementation, linear computational complexity in inference in both the number of samples and dimensions, robustness to small perturbations of the input data, out-of-sample ability, and a parameter-free inference. Yet, the obtained projections show relatively poor performance in terms of standard projection quality metrics (Espadoto et al., 2021b) when compared to other state-of-the-art DR techniques such as t-SNE or UMAP. In terms of visual patterns (Sec. 2.5), these projections expose a starburst-like structure with thin and elongated clusters emerging from their center — which makes them use the available projection space far less efficiently than, for example, t-SNE, Isomap, or MDS. As such, autoencoders have been mainly used for non-visualization DR tasks such as feature extraction from large image or text data.

More recently, the Neural Network Projection (NNP) method has been proposed (Espadoto et al., 2019a). In contrast to autoencoders, NNP is a fully supervised method: Given a dataset $\mathbf{X}$ and user-selected projection technique P , NNP learns to imitate $P(\mathbf{X})$ using a simple fully-connected neural architecture. NNP yields surprisingly good results for its simple architecture, *i.e.*, can learn to reproduce virtually any projection technique

P using a training-set of roughly a few thousand samples. Its main drawback regards the quality of the produced projections: In terms of quality metrics, NNP scores 5% to 10% lower than the so-called training projection $P(\mathbf{X})$ — the higher the quality of $P(\mathbf{X})$, the harder can NNP imitate it. Visually, this is immediately noticeable as an increased fuzziness that mixes point groups which are otherwise well separated in the training projection (see Sec. 2.3.3; also see Fig. 2.5). While NNP can significantly accelerate the computation of projections — for example, it is thousands of times faster than classical t-SNE on datasets of hundreds of thousands of samples or more — its lower quality makes it a questionable alternative in practice.

Several attempts have been done to reduce the aforementioned fuzziness by variations of NNP (see for the following Fig. 2.5). Self-Supervised Neural Projection (SSNP, [Espadoto et al., 2021a](#)) adds label information, provided either via ground-truth annotations present in the dataset $\mathbf{X}$ or computed as pseudo-labels via clustering, to yield better cluster separation. However, SSNP cannot imitate the style of a user-selected projection technique P as NNP does; in this sense, SSNP is closer to a refined version of autoencoders. k-NNP ([Modrakowski et al., 2022](#)) refines NNP by projecting groups of neighbor samples rather than individual samples, thereby achieving a small performance increase (and fuzziness decrease). Further work explored the hyperparameter and design space of NNP and SSNP by considering alternative architectures, training procedures, and cost functions ([Espadoto et al., 2020](#); [Oliveira et al., 2023a](#)), with only small performance gains. Separately, NNP was extended to learn a projection technique P and the variation of its hyperparameters such as the perplexity p used by t-SNE ([Appleby et al., 2022](#)). Still, the results delivered by this approach suffer from the aforementioned fuzziness.

Sharpened Dimensionality Reduction (SDR) is a separate approach for computing projections with good cluster separation ([Kim et al., 2022b](#)). The core idea is quite simple: Given a dataset $\mathbf{X}$, one first ‘sharpens’ the distribution given by its points using a variant of mean-shift ([Comaniciu and Meer, 2002](#)), after which a fast and typically lower-quality projection technique can be used, *e.g.*, Landmark MDS ([De Silva and Tenenbaum, 2004](#)). The initial data sharpening counterbalances the relative quality lack of the cheap projection technique being used to yield reasonable cluster separation. Since mean shift is itself a costly operation, this approach has been sped up by deep learning the entire pipeline by a method known as SDR-NNP ([Kim et al., 2022a](#)). Overall, SDR-NNP yields similar results in terms of quality to NNP while incurring significant additional training costs due to the need to learn the data sharpening step.

Given the success of NNP for learning direct projections, it is not surprising that a nearly identical approach was used to learn inverse projection functions ([Espadoto et al., 2019b](#)). As NNP, this approach can handle any direct projection technique P chosen by the user. Inverse projections

are also learned by construction by autoencoder-based approaches such as SSNP. Given the computational scalability of such inverse projection techniques, they have been the immediate candidates for computing decision boundary maps (Oliveira et al., 2022, 2023b).

In our work in Chapter 3, we address the main limitation of NNP-class approaches, namely, their relative drop of quality and creation of fuzzy projections. Our approach exploits a different direction than the above-mentioned works, namely, the incorporation of priors that favor high-density point patterns in the projection which are separated by clear gaps. Our results show that we can preserve all desirable aspects of NNP-class methods while reducing the aforementioned fuzziness. Separately, in Chapter 4, we replace the standard autoencoders in SSNP by variational ones, and propose ShaRP, a technique that is able to control the shape of clusters created by the projection (see also Fig. 2.5).

The academy is not paradise. But learning is a place where paradise can be created. The classroom, with all its limitations, remains a location of possibility.

Teaching to Transgress
BELL HOOKS

3.1 INTRODUCTION

As Chapter 2 has extensively detailed, the exploration of high-dimensional data, which consists of samples having tens to thousands of dimensions, also called measurements or attributes, is one of the key challenges of modern information visualization and visual analytics. In the same chapter, we have introduced *dimensionality reduction* (DR) methods as being the main class of techniques that address this visualization task. As explained there, DR methods have several unique advantages — as opposed to other visualization methods — for exploring high-dimensional data. Most importantly, such methods scale well, both computationally and in terms of the use of the available visual space, with both the number of samples and dimensions. They also effectively transform data-space tasks such as *measuring* correlations, relationships, and naturally-occurring clusters, into *visual* tasks that take as input the visual patterns exhibited by the projection scatterplot. That is, high-quality projections map data patterns to visual patterns which can be interpreted by a human so as to infer properties of the corresponding data patterns¹.

As outlined in Secs. 2.3.3 and 2.7.4, across the entire family of DR methods, parametric projection methods, and in particular NNP-class methods (Espadoto et al., 2019a), have many advantages such as computational speed, simplicity, and out-of-sample ability. However, as also explained there, such methods come in with a large amount of diffusion — or lower quality — as compared to state-of-the-art non-parametric methods such as t-SNE or UMAP. This makes them non competitive with non-parametric methods despite the aforementioned advantages.

¹ This chapter is based on the paper “APPA: A Cluster-Preserving Approximating Parametric Projection Algorithm” (Machado and Telea, 2026).

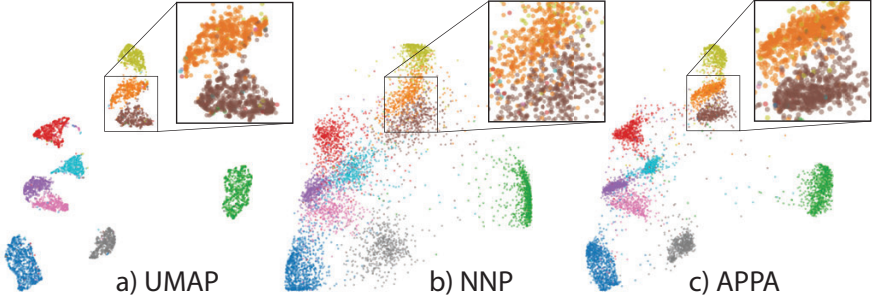


Figure 3.1: Diffusion effect of NNP. a) Projection of the USPS dataset (Hull, 1994) created by UMAP. b) The approximate projection created by NNP. Diffusion smooths out well-defined clusters in (a) and merges neighboring clusters. c) Our technique, APPA, shows reduced diffusion and well-separated clusters.

In this chapter, we propose APPA, standing for Approximating Parametric Projection Algorithm, a technique that strongly reduces the diffusion problem of NNP. APPA does so by using regularization techniques with the help of soft barrier functions, with only $O(n)$ extra computational cost in its training loop for n data points and *no additional cost* at inference time. Its structure is otherwise identical to NNP, meaning it inherits all of NNP’s desirable properties, *e.g.*, genericity, speed, out-of-sample ability, and ease of implementation and use. We make APPA’s Python code openly available (Machado and Telea, 2025). With the development of APPA, we address our RQ1.

The structure of this chapter is as follows. Section 3.2 introduces related work. Section 3.3 details both NNP and how we extend it that tackle NNP’s diffusion problem. Section 3.4 evaluates our method against several well-known projection methods, including NNP, on a variety of datasets and shows our method’s ability to compute high-quality, low-diffusion, projections. Section 3.5 discusses our method’s key features and concludes the chapter with future work directions. We provide additional information and experimental data in Appendix A.1.

3.2 RELATED WORK

Notations: We repeat here, for the reader’s convenience, the notations used further in this chapter (for full details, see Sec. 2.2).

We denote by $\mathbf{X}$ a dataset containing m data points $\mathbf{x}_i \in \mathbb{R}^n$ where $\mathbf{x} = (x_1, \dots, x_n)$. The projection of a single data point is denoted by $\mathbf{y}_i = P(\mathbf{x}_i)$, where $P : \mathbf{X} \rightarrow \mathbb{R}^q$ is any DR algorithm; the projection of the entire dataset is denoted by $\mathbf{Y} = P(\mathbf{X})$. P is computed so as to minimize

various types of costs, or errors, between $\mathbf{X}$ and $P(\mathbf{X})$. A neural network used to approximate P is denoted by $\Pi_\phi : \mathbb{R}^n \rightarrow \mathbb{R}^q$, where ϕ are weights and bias parameters. We denote the Euclidean norm, respectively L1 norm, of a vector by $\|\mathbf{x}\|_2 = (\sum_j x_j^2)^{\frac{1}{2}}$, respectively by $\|\mathbf{x}\|_1 = \sum_j |x_j|$.

Within the many existing DR algorithms, *parametric* techniques exhibit several important desirable properties (Sec. 2.3.3): These methods map the entire high-dimensional space $\mathbb{R}^n$, rather than a given dataset $\mathbf{X} \subset \mathbb{R}^n$, thus provide actual functions $P : \mathbb{R}^n \rightarrow \mathbb{R}^q$, where q indicates the dimensionality of the projection space (typically, $q \in \{2, 3\}$ for visualization purposes). As such, parametric methods are deterministic (yield the same output for the same input), robust to small changes of the input, and have the out-of-sample property (can project additional samples atop those created by an existing projection, thereby supporting the analysis of changing datasets).

Within the class of parametric DR techniques, deep-learning-based techniques occupy an important place. As outlined in Sec. 2.7.4, such methods, in particular the supervised ones promote by the NNP technique (Espadoto et al., 2019a) and refinements thereof, achieve nearly all desired properties for dimensionality reduction. NNP (and refinements thereof) are *projection-agnostic*, meaning they do not place any requirements on the projection P which is approximated. NNP’s backbone is a deep neural network, so it inherits the desirable properties of those: training is *fast* — due to the existence of libraries that can use GPU acceleration — and can be done using any off-the-shelf gradient-based optimizer. Inference is also parallelizable and fast, scaling *linearly* with the number of data points to be projected, as opposed to *e.g.* UMAP and t-SNE, which both scale super-linearly. Additionally, NNP builds a projection function that works on *unseen data* (also called out-of-sample data), whereas other DR algorithms require the complete recomputation of the projection if even a single new data point is added. NNP’s implementation is *simple* and requires *no parameters* to be tweaked with in inference mode. Last but not least, NNP’s neural network can be adapted depending on the data type one wishes to project — *i.e.*, convolutional networks for time series data or images and attention networks for text data.

However, NNP has a key drawback: It builds a projection function that causes well-defined point clusters in the training projection to spread out, or diffuse, over the projection space, reducing the inter-cluster distance and causing mixing of different data categories (see Fig. 3.1 and the earlier discussion on this topic for parametric projections in Sec. 2.3.3). We claim that this diffusion effect, illustrated in Fig. 3.1, is essentially due to overfitting (see next Sec. 3.3). This diffusion is highly undesirable since the resulting projections will convey a far poorer cluster separation than the actual training projection captures from its input data. The diffusion effect *worsens* when projecting out-of-sample data — while this is the key

use-case for NNP. Attempts to alleviate this problem, for example learning projections of neighborhoods instead of single points (Modrakowski et al., 2022), or tweaking the training hyperparameters (Espadoto et al., 2020; Appleby et al., 2022), have shown only very limited success — the diffusion phenomenon is still visible in their results. All in all, diffusion negates all other desirable properties of NNP and makes NNP unable to replace slower, but higher-quality, projection techniques such as t-SNE or UMAP.

We present a principled justification as to why NNP degrades in the out of sample regime. Further on, we present a new Approximating Parametric Projection Algorithm (APPA), that yields a measurable quality improvement in terms of learning to approximate any given reference multidimensional projection. We further develop a sample-based version of APPA, allowing control over the amount of desirable diffusion introduced in the learned projection. We demonstrate the abilities of APPA on a variety of datasets and ground-truth projection algorithms.

3.3 METHOD

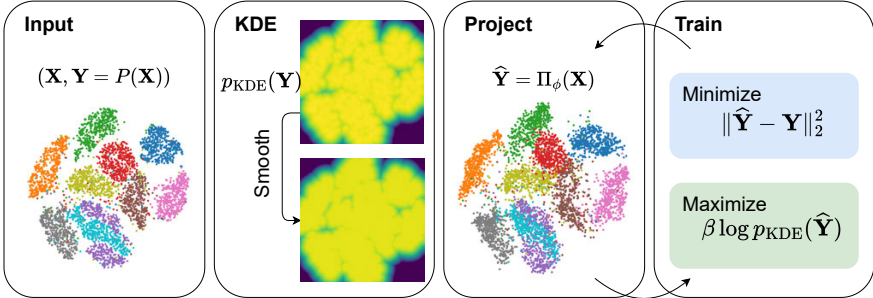


Figure 3.2: APPA’s architecture. APPA takes as input a dataset and its projection, computed by any method. APPA then builds a smooth KDE which determines the low (dark blue) and high (yellow) density areas in the projection. The approximating neural network is then fit to the reference projection with the additional goal of avoiding low-density areas. See Section 3.3.2.

Looking again at Fig. 3.1, we see that in dense regions of the projection space, NNP is reasonably accurate, so we do not want to strongly modify it there. Between clusters, however, NNP degrades and introduces diffusion. As such, we design our method APPA to *regularize* NNP by avoiding placing points in low-density regions in the scatterplot, as described next.

3.3.1 Neural Network Projection (NNP)

Espadoto et al. (2019a) proposed to use a neural network $\Pi_\phi : \mathbb{R}^n \rightarrow \mathbb{R}^q$ to parametrically approximate any reference projection computed on some dataset $\mathbf{X}$ by a DR technique of choice P . This method, called Neural Network Projection (NNP), idea is simple and quite effective: Given training data $\mathbf{Y} = P(\mathbf{X})$, NNP learns the parameters ϕ by optimizing the mean squared error loss

$$\mathcal{L}_{\text{NNP}}(\phi) = \sum_{i=1}^m \|\mathbf{y}_i - \Pi_\phi(\mathbf{x}_i)\|_2^2. \quad (3.1)$$

between the network’s predictions and the reference projection $\mathbf{Y}$.

NNP additionally uses *early stopping*, a simple form of regularization. Once the error on a held-out validation set stops decreasing, training is stopped. While valid, this technique is not enough to prevent overfitting, which is responsible for low projection quality when using the trained network Π_ϕ on unseen data. This quality loss is reflected both by lower quality metrics and visually by the ‘diffusion’ of point clusters that are well separated in the reference projection.

k -NNP (Modrakowski et al., 2022) aims to improve NNP by learning to project *neighborhoods* of points from the data space to the projection space. This is motivated by the observation that projections are inherently non-local: the position of each point depends on the position of possibly every other point in the projection — with influence diminishing according to distance between such points. While k -NNP marginally improves on NNP’s quality, it still fails to prevent the mentioned diffusion problem. Separately, a wealth of training heuristics, hyperparameters, and loss functions have been explored with NNP (Espadoto et al., 2020; Appleby et al., 2022). As for k -NNP, these only yielded marginal quality improvements.

3.3.2 APPA: avoiding low-density areas

NNP’s training procedure consists of fitting a regression model from $\mathbb{R}^n$ to $\mathbb{R}^q$. This can be understood as finding the maximum likelihood estimate (MLE) under a Gaussian conditional model

$$p_\phi(\mathbf{y}|\mathbf{x}_i) = \mathcal{N}(\mathbf{y}|\mu = \Pi_\phi(\mathbf{x}_i), I) \quad (3.2)$$

$$\hat{\phi} = \arg \max_{\phi} \log p_\phi(\mathbf{y}_i|\mathbf{x}_i) \propto \|\mathbf{y}_i - \Pi_\phi(\mathbf{x}_i)\|_2^2, \quad (3.3)$$

where the maximization is performed over the training set $(\mathbf{X}, \mathbf{Y} = P(\mathbf{X}))$.

While this works for learning *point-wise* relationships between data points $\mathbf{x}_i$ and their projections $\mathbf{y}_i$, it fails to consider how the projection algorithm P induces a *probability distribution* over the projection space. The

overall effect of P can be taken into account by introducing a *prior* and resorting to Maximum A Posteriori (MAP) estimation

$$\hat{\phi} = \arg \max_{\phi} \log p(\mathbf{y}|\Pi_{\phi}(\mathbf{x}_i)) + \log p(\Pi_{\phi}(\mathbf{x}_i)). \quad (3.4)$$

Note that NNP uses only the first term in the right hand side of Eqn. 3.4. The second term — that we introduce with APPA — constrains the projection, sharpening its shape by explicitly avoiding low-probability regions.

The choice of an appropriate prior is crucial for APPA’s success. We use a prior derived by Kernel Density Estimation (Rosenblatt, 1956), $p(\mathbf{y}) = \text{KDE}(\mathbf{Y})$, computed over the reference (training) projection $\mathbf{Y} = P(\mathbf{X})$. We use a Gaussian kernel with small bandwidth (0.01) since we want our prior to quickly decay away from the training projection. While non-parametric, this prior is differentiable with respect to its argument, so it is suitable for use in APPA’s loss function. We also replace NNP’s Gaussian likelihood by a Laplacian likelihood to further strengthen the desired sharpening effect. This amounts to replacing NNP’s Mean Squared Error loss with a Mean Absolute Error (L1) loss.

Putting all above together, APPA’s loss function to be minimized is

$$\begin{aligned} \mathcal{L}_{\text{APPA}} &= \sum_{i=1}^n [-\log p_{\phi}(\mathbf{y}_i|\mathbf{x}_i) - \beta \log p(\Pi_{\phi}(\mathbf{x}_i))] \\ &= \sum_{i=1}^n \|\mathbf{y}_i - \Pi_{\phi}(\mathbf{x}_i)\|_1 - \beta \log p(\Pi_{\phi}(\mathbf{x}_i)), \end{aligned} \quad (3.5)$$

where the first term in the summation is a regression error from the reference projection, and the second is a regularization term with weight β encouraging the distribution of the approximating projection to match that of the reference projection. Simply put, our training combines two objectives: in high-density regions, we reproduce the reference projection; in low-density regions, we regularize towards the reference projection’s shape. In all our experiments, we use $\beta = 0.002$.

To allow training to focus only on faithfully reproducing the projection in high-density areas, we clip the log-prior values to a maximum of -2 . This flattens high prior probability areas causing the gradient of the regularization term to vanish there. For this we use

$$\mathcal{L}_{\text{APPA}} = \sum_{i=1}^m \|\mathbf{y}_i - \mathbf{p}_i^{\phi}\|_1 - \beta \min(\log p(\mathbf{p}_i^{\phi}), -2), \quad (3.6)$$

where we denote $\mathbf{p}_i^{\phi} = \Pi_{\phi}(\mathbf{x}_i)$.

As Sec. 3.4.2 next shows, our loss function (Eqn. 3.6) produces slightly worse results than NNP’s loss (Eqn. 3.1) on the training set, as expected given our extra regularization — but strongly reduces diffusion on unseen data.

3.3.3 Sampled APPA

As explained in Sec. 3.3.2, APPA’s regularization term (Eqn. 3.6) introduces an effect opposite to diffusion, which we call *sharpening*. In practice, users may want to control the strength of this sharpening effect. This is difficult to do reliably in APPA. For this reason, we present next a Sampled version of APPA (SAPPA), which allows simple sharpening control.

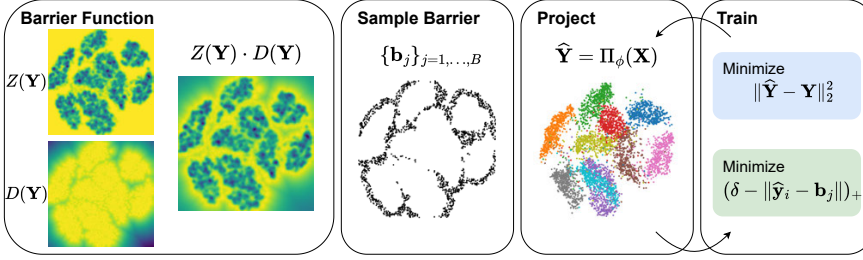


Figure 3.3: SAPPA’s architecture. This version of APPA uses an explicit set of barrier points $\mathbf{b}_j$. The algorithm minimizes the regression error with respect to the reference projection $\mathbf{Y}$ as well as a penalty linear in the distance of all points to the barrier. See Section 3.3.3 for further details on sampling the barrier and the proximity penalty.

SAPPA achieves the same effect as APPA — *i.e.*, avoid low-density areas — by sampling the projection space regions we would like to avoid and then explicitly penalizes closeness to such samples — see Fig. 3.3.

For this, we first compute a *barrier function* $\mathcal{B} : \mathbb{R}^q \rightarrow \mathbb{R}$ that combines the effect of two terms as follows. The first term $Z(\mathbf{y})$ disables the barrier close to points in the training projection $\mathbf{Y}$ and enables it slightly further away from such points; this prevents diffusion close to points in $\mathbf{Y}$. We compute Z as

$$Z(\mathbf{y}) = \frac{1}{1 + p(\mathbf{y})}, \quad (3.7)$$

where p is the KDE-based prior introduced in Sec. 3.3.2 and $\mathbf{y}$ is any point in the projection space $\mathbb{R}^q$.

The second term disables the barrier at points $\mathbf{y}$ that are far away from *all* points in the reference projection. This term is defined as

$$D(\mathbf{y}) = \frac{1}{1 + \min_{\mathbf{y}' \in \mathbf{Y}} \|\mathbf{y} - \mathbf{y}'\|}, \quad (3.8)$$

which we compute by using a nearest-neighbor algorithm. The rationale behind using D is that we do not want to explicitly constrain our projection in areas far away from the training projection $\mathbf{Y}$.

The final barrier is computed by multiplying the two desired effects as

$$\mathcal{B}(\mathbf{y}) = Z(\mathbf{y})D(\mathbf{y}) \quad (3.9)$$

Table 1: Datasets used for evaluation in this work. †: post-processed subsets of original datasets. See Appendix A.1.2 for additional details.

Name	Dims. (d)	Size (n)	# Classes
Cats and Dogs	2048	25 000	2
CIFAR10 [†]	1920	12 000	2
CIFAR100 [†]	1920	6 000	2
FashionMNIST	784	60 000	10
HAR	561	7 352	6
IMDB	500	25 000	2
MNIST	784	60 000	10
Reuters	5000	8 432	6
Spambase	57	4 601	2
USPS	256	9 290	10

We uniformly sample B on a 300×300 grid and keep only the 80% highest values, creating a set of barrier points $B = \{\mathbf{b}_j\}$. Using these barrier points, we construct the loss

$$\mathcal{L}_{\text{SAPPA}} = \sum_{i=1}^n \|\mathbf{y}_i - \mathbf{p}_i^\phi\|_2^2 + \beta \sum_j (\delta - \|\mathbf{p}_i^\phi - \mathbf{b}_j\|_2^2)_+, \quad (3.10)$$

where $\mathbf{p}_i^\phi = \Pi_\phi(\mathbf{x}_i)$ and $(x)_+ = \max(0, x)$. Simply put, this loss adds a linear penalty to points within a band of width δ from the training projection $\mathbf{Y}$, so limits diffusion close to $\mathbf{Y}$; and has no effect further away from $\mathbf{Y}$. Users can then control δ to tune the desired sharpening effect. In our experiments, we fix $\beta = 1.0$.

3.4 EVALUATION

We evaluate our approach for a variety of projection methods P — namely t-SNE, UMAP, Isomap, and PCA — across 10 datasets commonly present in DR literature benchmarks (see Tab. 1). We compare our results with NNP (Sec. 3.4.1) and see how (S)APPA strongly alleviates the diffusion problem. We further show that APPA yields better quality as measured by standard projection quality metrics (Sec. 3.4.2). Next, we explore how users can control sharpening via the parameter δ (Sec. 3.4.3). We also explore how (S)APPA behaves when projecting increasing amounts of data unseen during training (Sec. 3.4.4). Finally, we show how our APPA computationally scales to handle projecting large datasets (Sec. 3.4.5).

3.4.1 Generating projections

We design our evaluation as follows. Each combination of dataset $\mathbf{X}$ and projection P makes an experiment dataset $\mathbf{D} = (\mathbf{X}, \mathbf{Y} = P(\mathbf{X}))$. We split $\mathbf{D}$ into a training set $\mathbf{D}_T = (\mathbf{X}_T, \mathbf{Y}_T)$ and an evaluation set $\mathbf{D}_E = (\mathbf{X}_E, \mathbf{Y}_E)$, with $|\mathbf{D}_T| = \frac{1}{3}|\mathbf{X}|$ and $|\mathbf{D}_E| = \frac{2}{3}|\mathbf{X}|$.

We then train the approximation algorithms NNP, APPA, SAPPa on $(\mathbf{X}_T, \mathbf{Y}_T)$ for at most 300 epochs using the Adam optimizer (Kingma and Ba, 2015), compute their output $\hat{\mathbf{Y}} = \Pi_\phi(\mathbf{X}_E)$, and compare it to the unseen part of the projection ($\mathbf{Y}_E$).

Figure 3.4 shows the resulting projections (additional examples are present in Appendix A.1.1). We immediately see diffusion arising when using NNP, both on the training data $\mathbf{X}_T$ and, even more so, on the test data $\mathbf{X}_E$. For example, consider NNP’s output when trained on the t-SNE projection of the FashionMNIST dataset (Fig. 3.4, row 3 from top). In the reference projection, we see well-isolated sample groups such as the orange ones (A1) or yellow ones (B1). These strongly diffuse to mix with the other sample groups for both the training and testing NNP projection (markers A2, A3, B2, B3 in the image). For the (S)APPA training projection, these samples re-create the t-SNE training projection patterns almost exactly (markers A4, A6, B4, B6). For the (S)APPA testing projection, these groups become slightly more diffuse but still separate clearly from the adjacent groups in the projection (markers A5, A7, B5, B7).

(S)APPA’s ability to contain diffusion is even more visible when learning UMAP, a DR method known to produce *sharp* separation — small dense clusters separated by a lot of white space. For the HAR dataset (Fig. 3.4, row 2 from top), the reference projection shows two very clearly separated clusters of cyan samples (C1) and yellow-and-pink samples (D1). These clusters exhibit very strong diffusion in both NNP training and testing projections (markers D2, D3, C2, C3). (S)APPA’s training projection keeps these clusters almost identical to the reference projection (markers C4, D4, C6, D6) and creates minimal diffusion for the testing projection (markers C5, D5, C7, D7).

The trend of (S)APPA producing less diffusion than NNP, *i.e.*, better preservation of finer structures from the reference projections, holds across every single test case. Evidently, (S)APPA does not *completely* prevent the diffusion problem, for reasons already explained when outlining our design in Sec. 3.3: There is still a small amount of data that gets placed in regions that were *not* populated by the reference projection. This is due to the continuous nature of Π_ϕ : It is always possible to construct an $\mathbf{x}'$ such that it falls in an inter-cluster location in the learned projection. Separately, we see how (S)APPA cannot create *additional* separation which was not present in the reference projection — see the Cats & Dogs dataset projected with PCA in Fig. 3.4. Since PCA cannot separate well the two clusters (cyan and

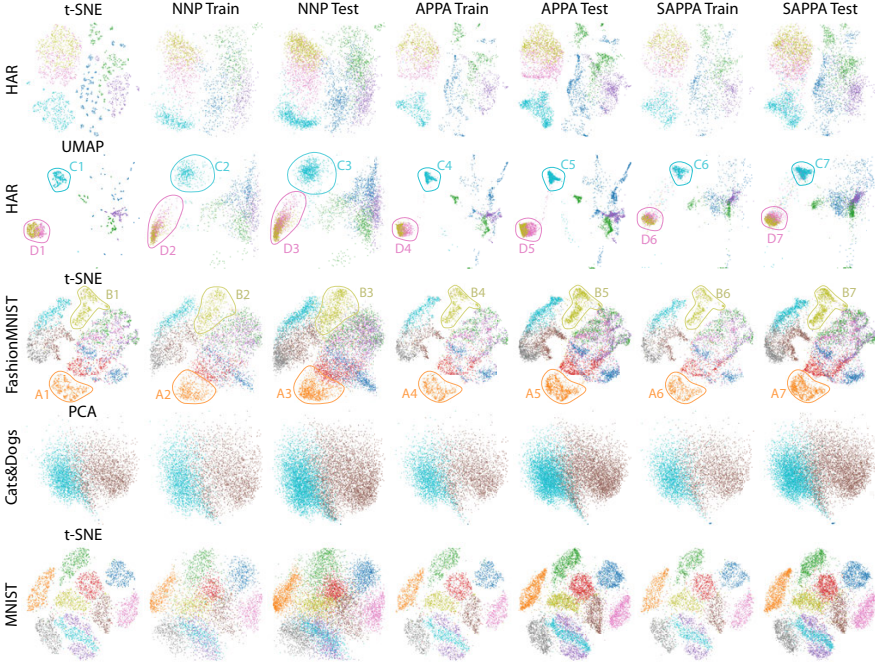


Figure 3.4: Using (S)APPA to reproduce different projections of different datasets. For each row we show, from left to right: the projection used for training; the result of running NNP on the training data and next on unseen data; APPA’s output on the training data and next on unseen data; and finally, SAPPa’s output on training data and next on unseen data. Overall, we see that (S)APPA better captures the training projection, and creates sharper clusters, than NNP.

brown points) due to its linearity, neither NNP nor (S)APPA can do this. Yet, even in this case we see that (S)APPA yields less diffusion than NNP.

3.4.2 Projection quality metrics

As is standard practice when evaluating projections, we further back up the visual evidence outlining (S)APPA’s higher quality in mimicking the reference projection (Sec. 3.4.1) by computing projection quality metrics for the compared methods and datasets. We use a total of 15 quality metrics, all common in DR literature, implemented by the library described in (Machado et al., 2025a). For all metrics that operate on k -neighborhoods, we use $k = 51$. All these metrics are detailed in Sec. 2.4. We present in the following results for a subset of these metrics, namely True Neighbors Rate, Trustworthiness, Continuity, Neighborhood Hit, and Stress — as these are

the most frequently used metrics in DR evaluation in the literature. We present results for the remaining metrics in Appendix A.1.3.

Figure 3.5 shows these metrics computed for all reference projections and datasets described in Sec. 3.4.1. In general, the reference projection method yields best values as expected — an *approximation* method, like NNP or (S)APPA, will likely not exceed this baseline. (S)APPA metrics come next, practically equal to each other and just slightly lower than the reference values. NNP metrics are consistently lower than (S)APPA ones indicating what we qualitatively saw so far, *i.e.*, that (S)APPA is a better approximator. Indeed, if diffusion is reduced, then a projected point’s neighbors are more often correct (True Neighbors, Continuity, Trustworthiness); those neighbors more often share the same class (Neighborhood Hit); and distances are better preserved (Stress). Conversely, if a projection becomes diffuse, like in the NNP case, then neighborhoods break apart and/or lose accuracy (impacting True Neighbors, Continuity, Trustworthiness). Diffusion also causes points that should be close together to shift away from each other. If the reference projection had good Neighborhood Hit, this shifting will lower its value, since previously well-defined neighborhoods start mixing. Distance Consistency also goes down in diffuse projections: If classes spread around the projection space, their centroids tend to be potentially close to each other. Finally, diffusion also increases Stress, as points become more uniformly distributed in the projection space, something that unlikely happens in data space due to the curse of dimensionality.

Figure 3.6 shows the accuracy of the approximate projections *vs* the projection they are trained to approximate, measured as Mean Square Error (MSE), both on training data $\mathbf{Y}_T$ and on the held-out data $\mathbf{Y}_E$. We see that, during training, both APPA and SAPPa yield MSE values (close to) zero for most points. This is evidence that using a prior to shape the learned projection — be it an explicit barrier function or a KDE directly, see Sec. 3.3 — allows training to focus on placing points only in regions of interest. In contrast, NNP produces such low-error projected points about half as often and also yields higher MSE values overall. On unseen data, all algorithms yield slightly larger MSE values than on training data. Yet, (S)APPA still yields lower MSE values than NNP, which numerically confirms the diffusion effect we saw.

Table 2 summarizes the MSE errors computed over all the tested dataset-projection combinations. We see that (S)APPA yield practically identical values which are significantly better than those given by NNP.

3.4.3 Controlling projection sharpening

As shown so far, both SAPPa and APPa substantially decrease diffusion. In particular, SAPPa does this explicitly by linearly penalizing placing

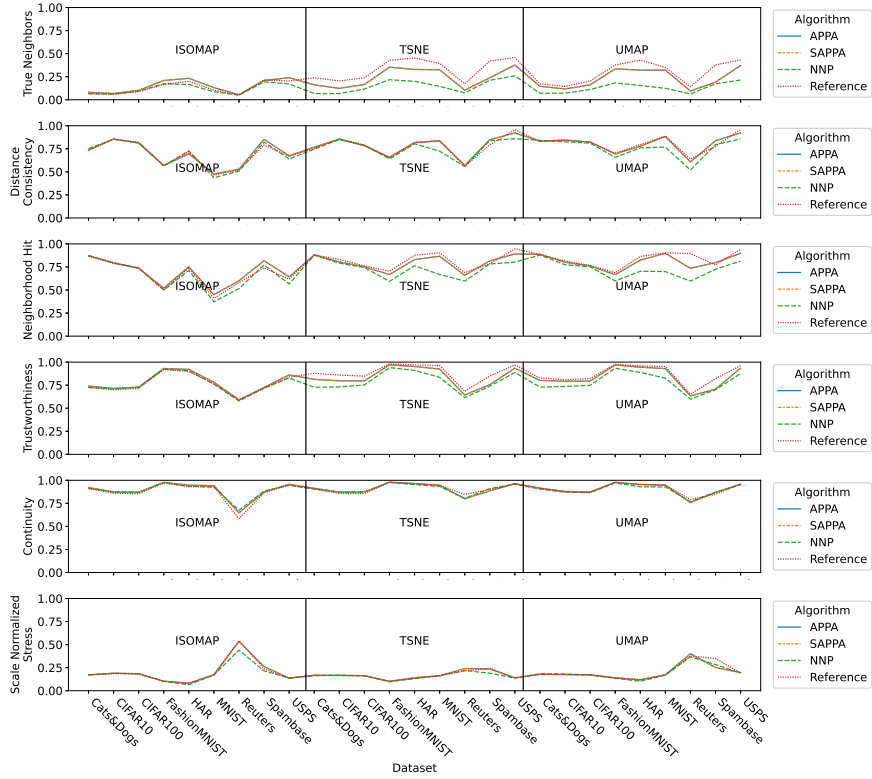


Figure 3.5: Projection quality metrics for (S)APPA and NNP when approximating different projection techniques. ‘Reference’ gives the value of each quality metric for the approximated projection. We see a clear “sandwiching” pattern: the reference projection has best quality metric values, followed by APPA, then by NNP.

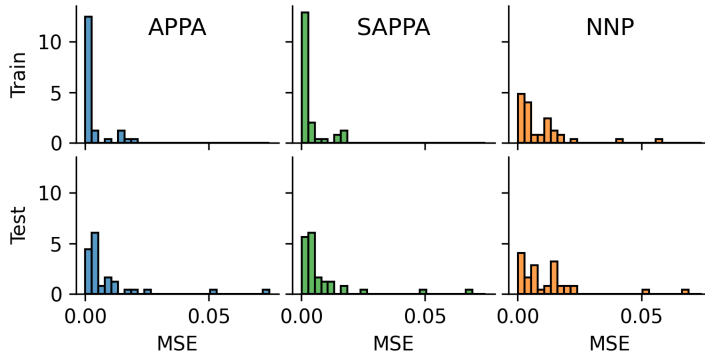


Figure 3.6: Mean Squared Error measured over training and test data for APPA, SAPPa, and NNP. Each plot shows the distribution of projected points per MSE value.

Table 2: Mean Squared Error with respect to reference projections for APPA, SAPPa, and NNP, averaged over all studied datasets and projections. Lower is better.

	APPA	SAPPa	NNP
Train	0.0032	0.0032	0.0088
Test	0.0086	0.0079	0.0112

points closer than a threshold δ to any barrier point $\mathbf{b}_j$ (see Sec. 3.3.3). In detail, the penalty term for a single projection point $\mathbf{y}_i$ and barrier point $\mathbf{b}_j$ in Eqn. 3.10 is

$$(\delta - \|\mathbf{y}_i - \mathbf{b}_j\|_2)_+ = \begin{cases} 0, & \text{if } \|\mathbf{y}_i - \mathbf{b}_j\|_2 \geq \delta \\ \delta - \|\mathbf{y}_i - \mathbf{b}_j\|_2, & \text{otherwise} \end{cases} \quad (3.11)$$

Hence, we can see δ as a *sharpness* parameter: The more we increase it, the farther away from the barrier points SAPPa must place its learned projection, yielding increasingly more concentrated point clusters from the training projection.

Figure 3.7 explores this effect. At $\delta = 0$, SAPPa reduces to NNP — there is no barrier-related penalty. We see slight sharpening with $\delta = 0.02$, with fewer points falling in the inter-cluster spaces in the projection. For $\delta = 0.04$, clusters become overall thinner or more densely packed. If we increase δ too much ($\delta = 0.1$), clusters collapse and merge, which is undesirable. In our experiments, $\delta = 0.02$ produces good sharpening results. Using δ thus allows analysts to easily control the overall sharpening effect they want to achieve in the learned projections.

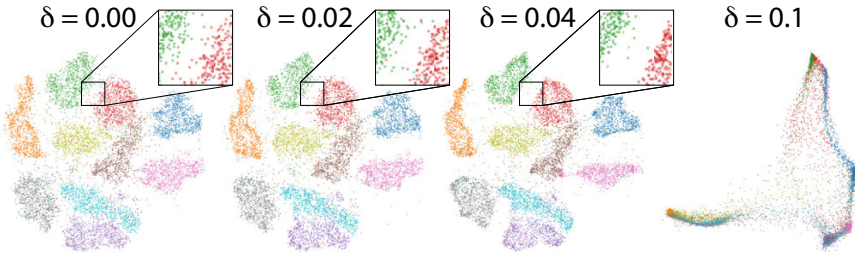


Figure 3.7: Setting the δ parameter in SAPPa for a t-SNE projection of the MNIST dataset. The higher δ is, the farther away from the barrier SAPPa learns to place its projection. Increasing δ sharpens the projection, up to where the projection becomes degenerate ($\delta = 0.1$).

3.4.4 Projecting unseen data

As already mentioned in Sec. 3.2, diffusion is an artifact, we claim, of NNP *overfitting* to its training data. This matches the finding that NNP degrades strongly for data not seen during training — the network is not able to generalize well to new data. Moreover, NNP degrades further as it is asked to project increasingly more data points that were not seen during training (see Fig. 2.2).

One can avoid this issue if, instead of approximating P — where P is any projection algorithm — we simply re-ran P on the new data. Two main problems exist with this. First, this is highly expensive if the underlying P is itself expensive as in the case of *e.g.* t-SNE. Secondly, many such algorithms (among which t-SNE) produce *unstable* projections due to their stochastic nature. Hence, visualizing a sequence of such projections showing increasingly more data (even if drawn from the same distribution) will not allow analysts to uncover any meaningful insights.

(S)APPA strikes a good trade-off between NNP’s ability to quickly produce an approximate projection for unseen data (but with high diffusion) and the exact projection P ’s ability to produce a projection with high quality (but in a costly and unstable manner). Figure 3.8 illustrates this by showing projections obtained via several methods for an increasingly large subset of the MNIST dataset. We use two different training projection techniques: t-SNE and UMAP. We then train NNP, APPA, and SAPPa to approximate each projection, using 5K training samples. We next use each of t-SNE, UMAP, as well as the three approximations of each (NNP, APPA, SAPPa) to project a growing number of unseen data points. For reference, the black-outlined images show the training projections. The results show how both t-SNE and UMAP yield well-separated clusters (in line with the known cluster separation in MNIST) but also high projection instability. At the other end of the spectrum, NNP shows very good stability — the same clusters get projected in the same areas — but *increasing* diffusion as the sample count increases. The latter is clearly a misleading artifact of NNP — we know, for this dataset, that drawing more samples does not create more mixed clusters. (S)APPA strike a good balance in the middle: Its projections show identical stability to NNP; and diffusion increases far less than NNP as the sample count increases.

3.4.5 Computational scalability

An approximating projection algorithm, such as (S)APPA or NNP, is only useful if it increases computing speed as compared to the reference DR algorithm it is trained on. To assess this, we compared two such DR algorithms (t-SNE and UMAP) with both NNP and (S)APPA for datasets ranging from 500 to 250K samples. In all cases, training used a maximum

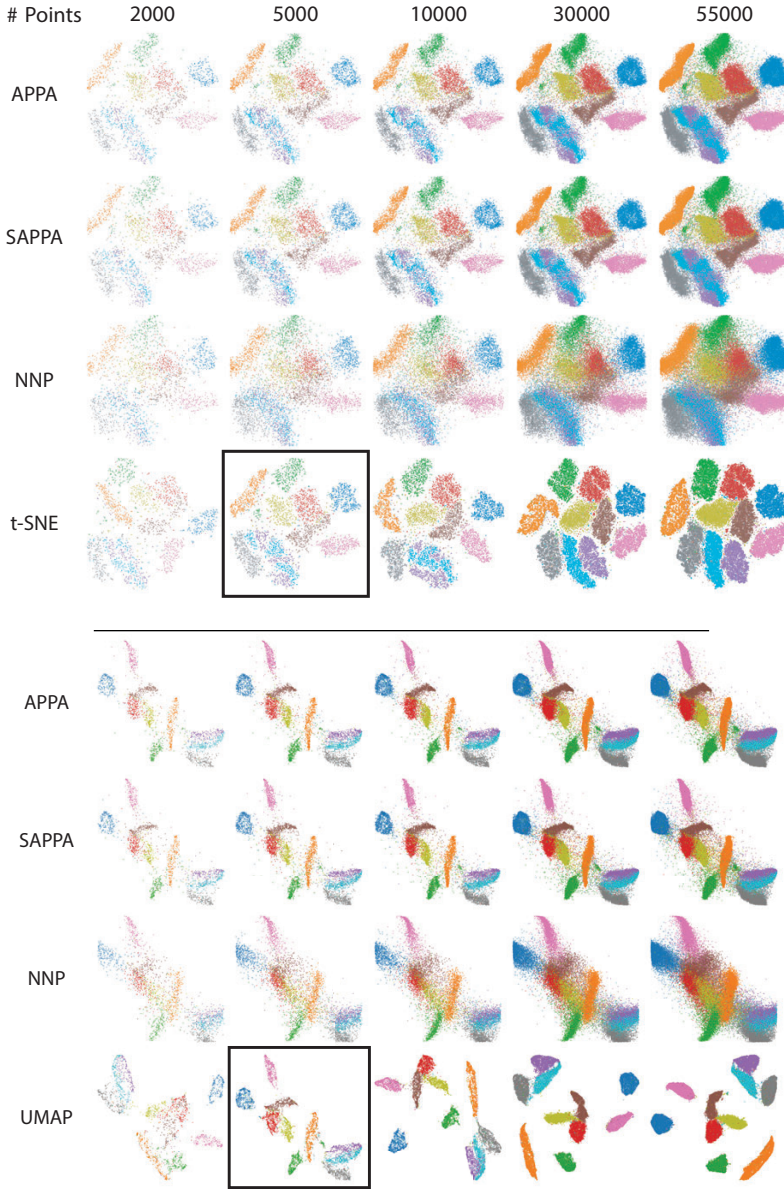


Figure 3.8: We train NNP and (S)APPA on a reference projection (t-SNE and UMAP, marked by black border). We then compare re-running the full projection algorithm (bottom row in each group) to running NNP (third row) and both versions of APPA (first two rows) for a dataset unseen in training. Re-running the full projection yields instability; re-running NNP yields strong diffusion. (S)APPA yields stability and significantly lower diffusion.

of 300 epochs and 5K training samples. Figure 3.9 shows the results. For growing dataset sizes, we see a strong increase in run time for t-SNE and UMAP. NNP runs the fastest of all due to its use of early stopping; yet, as discussed in Secs. 3.4.2 and 3.4.4, it produces lower quality, diffuse, projections. APPA’s training is roughly two times slower than NNP since it does not use early stopping. SAPPa is roughly 20% slower than APPA for small datasets due to the explicit computation of distances to the barrier (Sec. 3.3.3). However, once trained, (S)APPa is as fast as NNP.

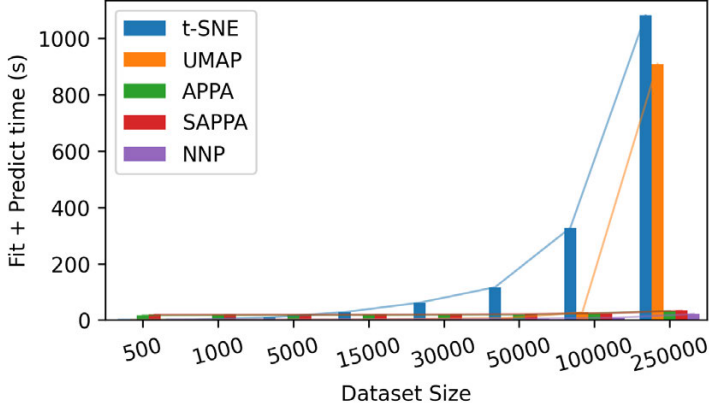


Figure 3.9: Computational speed of (S)APPa, NNP, t-SNE and UMAP. We train (S)APPa and NNP for at most 300 epochs with 5K samples. Each value is averaged over 3 runs.

APPA and SAPPa are both twice as slow as t-SNE for datasets of up to 5K samples. For 15K samples, they are on average as fast as t-SNE. APPA and SAPPa have about the same speed as UMAP for datasets with 100K samples, being slower for all smaller datasets. However, for the largest dataset size we experiment with — 250K samples — SAPPa and APPa are between 25 and 30 times *faster* than t-SNE and UMAP. More importantly, the *trends* in this chart are telling: t-SNE and UMAP costs increase superlinearly with the sample count while NNP, SAPPa, and APPa are linear with this count *and* have a quite low slope. All in all, this means that APPA and SAPPa practically offer the same significant speed-ups *vs* classical projection algorithms as NNP but, as shown earlier, with increased quality.

3.5 DISCUSSION AND CONCLUSION

In this chapter, we have presented APPA and its sample-based version, SAPPa, two techniques that learn to approximate dimensionality reduction algorithms in ways that combine several desirable features. Compared

to NNP, our closest competitor that we improve upon, (S)APPA cover the following points:

Genericity: As NNP, (S)APPA can learn to mimic any DR technique in a black-box fashion, that is, irrespective of the data dimensionality, nature of the data, or internals of the learned DR technique.

Quality: (S)APPA significantly improves upon NNP’s weakest point, namely the latter’s tendency to generate diffusion effects in projections of otherwise well-separated samples. Both visual comparisons and study of projection quality metrics show that such improvements consistently happen for all studied datasets and learned DR techniques. Quality increase is done in a principled fashion, *i.e.*, by regularizing the projection via the addition of a simple cost term that favors the creation of point distributions similar to those present in the training projection. This removes, for the first time to our knowledge, the key weakness of NNP.

Out of sample: As NNP, (S)APPA can project changing datasets in a stable manner, *i.e.*, by placing the same samples at the same locations in these multiple projections. However, while NNP shows increased diffusion as such datasets grow, (S)APPA keeps stability and only adds minimal diffusion to the process.

Ease of use: As NNP, once trained, (S)APPA requires no parameters to be set, making its use immediate. During training, SAPPa proposes one extra parameter δ that models the amount of sharpness of the resulting projections, which comes with a given preset ($\delta = 0.2$).

Scalability: As NNP, (S)APPA is very fast to train, yields good training results with about 5K training samples on all tested examples, and is linear in sample count in inference (projection) mode. While (S)APPA’s training is roughly two times slower than NNP, its inference cost is identical to NNP, which is orders of magnitude smaller than expensive DR methods such as t-SNE. This makes (S)APPA an attractive method for projecting datasets of millions of samples.

Applications: As (S)APPA provides a more accurate drop-in replacement for NNP, it can be used in any applications where NNP-like approximations are relevant. For example, NNP’s desirable properties have recently motivated its use in fields beyond DR such as Graph Drawing ([Hartskeerl et al., 2025](#)). (S)APPA can be directly used to improve such applications.

Limitations: (S)APPA works well to reduce diffusion if the reference projections have good cluster separation. If the reference projection places points close to each other without much inter-cluster distance, our regularization

will not be as effective in reducing diffusion (see *e.g.* the PCA example in Fig. 3.4). Additionally, even if clusters are well-separated, they might consist of too few points. This makes the KDE p in Eqn. 3.6 too sharply peaked, with near-zero gradients almost everywhere, which hinders regularization effectiveness. One way to fix this — to be explored in future work — is by oversampling such small clusters according to the KDE, akin to a bootstrapping process.

We see several directions of future work. SAPPa’s training can be easily accelerated by using computationally-efficient distance transform methods, which are linear in the resolution of the generated image, and can be implemented both on the CPU (Meijster et al., 2002) or on the GPU (Cao et al., 2010), to compute the barrier, even on higher resolutions than our current 300^2 pixels, yielding finer-grained sharpening control (Eqn. 3.9). Adaptive diffusion can be incorporated to *e.g.* model true dataset distances in the projection while keeping clusters sharp where appropriate. Finally, (S)APPa can be easily extended to efficiently and accurately project time-dependent datasets, an area where few methods exist in the DR literature (Vernier et al., 2020a).

(S)APPa demonstrate how to sidestep a fundamental issue in its predecessor projection approximation methods. This removes an important barrier to the widespread application of approximate projections, avoiding the scalability issues of traditional projection methods. However, (S)APPa operate on a reference projection, meaning they are *supervised* learning algorithms — *i.e.*, they perform regression. We next investigate how to build projections without any reference using Neural Networks in Chapter 4. What kind of architecture is adequate to solve this problem? What sort of structure can we portray in such a projection? What new capabilities does such a projection algorithm possess?

PROJECTION WITH CONTROLLABLE VISUAL SIGNATURE

Que educador seria eu se não me sentisse movido por forte impulso que me faz buscar, sem mentir, argumentos convincentes na defesa dos sonhos por que luto?[†]

Pedagogia da Esperança
PAULO FREIRE

Projection techniques aim to preserve as much as possible of the so-called *data structure* by mapping data structures to visual structures. In turn, analysts seeing such visual patterns in a projection scatterplot can infer the existence of the corresponding data patterns. As discussed in Chapter 3, these visual patterns are crucial for the usability of a projection algorithm or approximation thereof. However, as we explained in Sec. 2.5, no single projection technique can map all possible data patterns of potential interest in some dataset to identifiable visual patterns. Separately, as explained at the same point, different projection techniques tend to create particular patterns which are not due to the input data but the cost function and optimization thereof they use¹.

The presence of such technique-specific and data-independent patterns, which we call *visual signatures* of projection techniques, can confuse analysts in misleading them for actual data patterns. However, controlling such visual signatures is hard with current projection methods, as these are designed around cost functions which do not incorporate measures on, or controls of, the visual patterns they create.

In this chapter, we address this problem by presenting **Shape Regularized Neural Projection (ShaRP)**, a novel projection method. ShaRP achieves visual-pattern control by extending the earlier autoencoder-based SSNP projection technique (Espadoto et al., 2021a) by a variational autoencoder that allows constraining the shapes of created visual patterns. ShaRP scales well with both dimension and sample count, works both in a supervised way (by class labels) and self-supervised way (by pseudola-

¹ This chapter is based on the papers “ShaRP: Shape-Regularized Multidimensional Projections” (Machado et al., 2023) and “Controlling the scatterplot shapes of 2D and 3D multidimensional projections” (Machado et al., 2024e).

[†] “What kind of educator would I be if I did not feel moved by a powerful impulse to seek, without lying, convincing arguments in defense of the dreams for which I struggle?”

bels created by data clustering), generically handles any quantitative high-dimensional dataset, and is simple to implement. ShaRP also provides, by design, an inverse projection mapping useful for interpolating new data space points and generating Decision Boundary Maps (Rodrigues et al. 2018, see Section 4.3.5). Finally, in contrast to earlier DR techniques that project data either in $\mathbb{R}^2$ or $\mathbb{R}^3$, ShaRP can also project data into the non-Euclidean space $\mathcal{S}^2$ — the surface of a 3D ball — which provides novel ways to explore high-dimensional data. This contribution provides an answer to our RQ2.

The structure of this chapter is as follows. Section 4.1 presents the preliminaries based on which our shape-controlled projection is designed. Section 4.2 details the ShaRP technique. Section 4.3 evaluates ShaRP by studying the effect of its hyperparameters and also comparing its results with other well-known projection techniques on a variety of datasets. Section 4.4 discusses our proposal. Finally, Section 4.5 concludes with future work directions.

4.1 PRELIMINARIES

4.1.1 Notations

For the reader’s convenience, we repeat here the notations introduced in Sec. 2.2 and used next. A dataset $\mathbf{X} = \{\mathbf{x}_i\}_{i=1,\dots,m}$ has m samples $\mathbf{x}_i = [x_{i1}, \dots, x_{in}]^T$, where $\mathbf{x}_i \in \mathbb{R}^n$ has components x_{ij} , $1 \leq j \leq n$ and an optional label $y_i \in \{1, \dots, K\}$. We use capitals to denote the set of all elements for the corresponding small letter, e.g., $\tilde{Y} = \{\tilde{y}_i\}_{i=1,\dots,m}$ is the set of (pseudo-)labels used for training. Variables marked with a hat (e.g., $\hat{\mathbf{X}}, \hat{Y}$) represent predictions produced by ML model. We denote the Euclidean norm by $\|\mathbf{x}\| = \sqrt{\mathbf{x}^T \mathbf{x}}$ and the expected value of a function of a random variable $\mathbf{z}$ distributed according to p by $\mathbb{E}_{\mathbf{z} \sim p}[f(\mathbf{z})]$. We use θ to denote probability distribution parameters, for example $\theta = (\boldsymbol{\mu} \in \mathbb{R}^2, \boldsymbol{\sigma}^2 \in \mathbb{R}^2)$ for a 2D Diagonal Gaussian distribution.

Projection algorithms (Sec. 2.3) can be defined as functions

$$P_\eta : \mathbb{R}^n \rightarrow \mathbb{R}^q, \quad (4.1)$$

where $q \ll n$ is the dimension of the created scatterplot (typically $q \in \{2, 3\}$) and η denote P ’s (hyper)parameters. Inverse projections (Sec. 2.6) are denoted as

$$P^{-1} : \mathbb{R}^q \rightarrow \mathbb{R}^n. \quad (4.2)$$

4.1.2 Controlling visual signatures

In related work (Sec. 2.4), we have presented a detailed overview of different projection quality metrics (PQMs) and explained how these capture several types of desirable properties of projections. In the following (Sec. 2.5), we argued that, while PQMs gauge *necessary* properties of projections, they are not *sufficient* in capturing the presence (or absence) of visual patterns that such projections can exhibit. This is an important aspect that underlies the work on shape-regularized projections presented in this chapter. As such, we elaborate this limitation of classical PQMs further below.

For something — a point set, in our case — to form a visual pattern, it needs to have a particular *shape* that distinguishes it from other point sets. In turn, this implies that patterns have a non-zero size (computed as some function of the distances between their points). More precisely, visual patterns can occur on given *scales* or ranges of sizes. The more complex a pattern’s shape is, the more points that pattern needs to contain — so that shape is realizable — and, arguably, the larger is the scale on which that pattern appears.

We argue next that these properties of visual patterns conflict with the aspects used in the design of common projection quality metrics. For this analysis, it is useful to consider the *scale* on which such PQMs operate, as follows:

- **point pair:** Metrics that characterize individual point pairs, such as stress, operate on a too fine scale to capture the shape of patterns; indeed, two points are too few to create a (complex) shape;
- **class:** Metrics that characterize all points of a given class (having a given label) are, in contrast, too coarse to capture pattern shapes. In theory, they can capture the shape formed by all points having the same label. However, in most datasets, such points do not form *together* a single, simple, pattern. Rather, they organize into many smaller groups, each with its own pattern, where each such group encodes a sub-variety of samples falling under the given class. This is not surprising given that projection method typically do *not* use labels in their operation; rather, they create scatterplots $P(\mathbf{X})$ that reflect only the data attributes of $\mathbf{X}$; labels are typically post-impressed on $P(\mathbf{X})$ to see how they correlate with the data to analyze *e.g.* classification ease (Rauber et al., 2017; Benato et al., 2024a). Hence, visual patterns exist typically at finer scales than classes;
- **global:** Computing single metrics for an entire scatterplot $P(\mathbf{X})$ is clearly too coarse to capture any visual pattern of interest or visual signature of a projection technique;

- **neighborhood:** Of all discussed scales, neighborhoods are the best suited to capture visual patterns. Yet, current PQMs typically define such neighborhoods in terms of the k -nearest neighbors of a point. There are two issues with this. First, setting k to different values will yield different quality judgments on the *same* projection (Martins et al., 2015). There is no simple way to determine how to set k to capture specific visual patterns. Most studies concerning this aspect limit themselves to suggesting low, fixed, values for k , e.g., $k = 8$ in Espadoto et al. (2021b), which is fine for consistent PQM computation across experiments but is a too low value to capture any significant pattern shapes; or mention that ‘good’ ranges to set k in (e.g., 40 to 60 in (Wattenberg et al., 2016)), which does not go beyond suggesting a good practical heuristic. Secondly, k -nearest neighborhoods can only model ‘circular’ patterns based on the distance to a central point. Patterns such as Isomap’s line-like structures, or autoencoders’ starburst-like shapes, mentioned in Sec. 2.5, cannot be in general modeled by k -nearest neighbors.

Summarizing the above, we see that projection-quality metrics, while useful in gauging various desirable aspects of a projection at different scales, cannot really capture the *patterns* that constitute a projection’s visual signature. As such, they cannot be used to design cost functions to actually *drive* a projection to generate specific visual signatures. With these limitations, several works have addressed the measurement — and only very partially, the control — of visual signatures of projections, as follows.

Initial work on perception-driven DR (Etemadpour et al., 2015; Wang et al., 2018) aims to factor the humans’ perceptual capabilities in the DR computation and quality assessment. Yet, developing quality metrics that target a variety of *visual patterns* is nontrivial, since these heuristics must simulate the humans’ complex pattern recognition process. Moreover, even if we possessed such a visual quality metric, it is not evident what would be a suitable *ground truth* to compare it to, since that ground truth has to somehow measure the patterns present in the data. More importantly, our goal here is different: while perception-driven DR aims to measure how users *see* patterns in a projection scatterplot, we aim to allow users to *control* how that scatterplot will look like in terms of its patterns. As such, perception-driven DR research can come after ours — to assess how users perceive our pattern-controlled projections — but not precede or replace it.

Cutura et al. (2022) use space-filling curves to adapt the position of data points in image thumbnail scatterplots such that they are non-overlapping. This can be seen as a local way to control the generation of visual patterns in a projection. Their idea is effective but limited to image datasets, while our proposed technique is designed to be generic.

Abbas et al. (2019, 2021) proposed a visual quality metric that mimics the users’ ability to distinguish one or more clusters in a scatterplot. The main differences of our work to theirs is that we (a) do not aim to propose a technique to create separated clusters, but to enforce the *shapes* of such clusters; and (b) our ground truth is not user perception, but the data space.

The perplexity parameter in t-SNE controls, among other things, a projection’s the visual appearance. Its effect is intricately enmeshed within t-SNE, leading to cluster shapes, sizes, and distances that do not necessarily convey meaning (Wattenberg et al., 2016). Our technique minimizes the variability of the projection’s visual appearance. Related to this, since fully doing away with hyperparameters might be infeasible, techniques such as HyperNP (Appleby et al., 2022) learn to simulate their effects on the resulting projection. This helps users quickly scan different perplexity values to choose the one generating a projection with the desired visual signature. In contrast to HyperNP, our technique uses a *single* set of hyperparameters to generate user-chosen visual patterns, reducing the need for such costly simulations.

Finally, Makhzani et al. (2015) propose an approach similar to ours. They adapt an auto-encoder into an adversarial setting to increase the quality of the projection (e.g., better cluster separability) that the vanilla auto-encoder would create. Our method instead aims mainly to put cluster shapes under the user’s control. In other words, they optimize a general-purpose quality metric, but do not control the visual patterns explicitly; we control these patterns explicitly, and measure a posteriori what the quality price incurred by the control is.

We believe that users can benefit from having *direct* control over the visual signatures of a projection technique for several reasons.

Firstly, the possibility of *picking* a visual signature for a projection gives users *direct* control over what is thus far an emergent property implicitly determined by a DR algorithm’s design and not necessarily a portrayal of true data patterns. For example, assume we see elliptical visual patterns (clusters) in a projection. With current projection techniques, we do not know if these reflect actual (elliptical) patterns in the *data* or they are produced by the projection technique. In contrast, our method allows controlling such patterns — so, when seeing them, we know that they are due to the projection and not the data; in other words, we argue that precise visual signature control is better than no control. To refine this example: In cases where users employ pseudolabels in their pipeline, outputting shapes that match the pseudolabeling algorithm’s assumptions would ensure a “match” between high and low dimensional space — for instance, producing round shapes when using *k*-means, which assumes isotropic Gaussian clusters).

Secondly, controlling the generated visual patterns can help in creating 2D or 3D scatterplots which better match a given interactive exploration

task at hand. For example, when performing interactive data labeling using rectangular selections or displaying image thumbnails over data clusters (see Fig. 4.3), a projection whose clusters resemble rectangles would be more suitable than one creating various-shaped clusters (modulo other aspects of the two projections, *e.g.*, quality).

4.2 SHARP: SHAPE-REGULARIZED NEURAL PROJECTION

We now present ShaRP, our novel DR technique, which is based on deep neural networks. Such networks can, in general, approximate complex non-linear functions and have several desirable features that ShaRP inherits:

Scalable: ShaRP scales linearly in the sample count because it avoids pre-computing pairwise distances or covariance matrices, like in PCA or t-SNE, and lends itself to hardware-acceleration through GPUs or TPUs tailored for fast deep learning.

Parametric: ShaRP operates in a “learn once, project as needed” fashion. It learns to parameterize a projection function instead of only outputting the projected points, such as t-SNE or UMAP. This allows ShaRP to project data it was not trained on along with existing data (out-of-sample ability).

Generic: ShaRP handles any dataset comprised of numeric features and can be applied to a wide range of datasets using the same or only slightly adapted hyperparameter settings.

Sound: ShaRP scores comparably to state of the art techniques in relevant projection quality metrics.

To these, ShaRP adds two flavors of **Shape Regularization**:

- Intra-projection: ShaRP creates point clusters having shapes coming from the same (user-controlled) *family*: ellipses, rectangles, triangles.
- Inter-projection: Running ShaRP over different datasets produces a consistent *visual signature* where differences in the projections are driven mainly by data patterns.

Invertible: Since ShaRP uses deep learning, it can be directly used to invert the projection mapping P to compute inverse projections P^{-1} . We show in Sec. 4.3.5 how we can use this ability to compute decision maps to explore the behavior of trained classification models.

ShaRP is implemented in Python using Keras (Chollet et al. 2015, Tensorflow backend) and Tensorflow Probability (Dillon et al., 2017) for sampling and calculating log-probabilities under different distributions. It is publicly available at <https://github.com/amreis/sharp>.

trates this for the well-known MNIST dataset. Yet, as the figure also shows, SSNP collapses some clusters into elongated shapes, which we argue is (a) unnatural, as it suggests some anisotropy in the sample distribution; (b) space-inefficient, as much white space is not used to depict data; and (c) suboptimal for visualization as we cannot *e.g.* easily select a cluster by rubberband tools or efficiently annotate it with a rectangular icon.

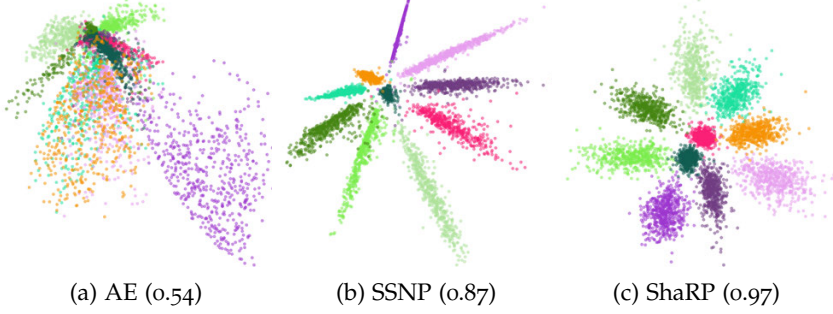


Figure 4.2: Comparison of projections of the MNIST dataset learned using (a) Auto-encoders, (b) SSNP (Espadoto et al., 2021a), and (c) ShaRP. SSNP and ShaRP were trained using the *ground truth* labels as class information — encoded, here and next, by colors. Values in brackets are Distance Consistency scores (higher is better, see Sec. 2.4.2).

ShaRP overcomes these shortcomings of SSNP by an explicit user-controlled shape regularization mechanism, described next (see also Section 4.2.2 for examples). ShaRP replaces SSNP’s Auto-Encoder (AE) with a Variational AE (VAE, Kingma and Welling, 2014). The key AE-VAE difference is the latter’s use of a *sampling* process in the network’s bottleneck layer. This, coupled with a necessary KL divergence (KLD) regularization term

$$\mathcal{L}_{\text{reg}}(\theta) = D_{\text{KL}}(q_{\theta} || p) \doteq \mathbb{E}_{\mathbf{z} \sim q_{\theta}} [\log(q_{\theta}(\mathbf{z}) / p(\mathbf{z}))] \quad (4.4)$$

has as an immediate effect on the regularization of the learned latent space: Using $\mathcal{L}_{\text{reg}}$ pushes the learned probability distributions q_{θ} toward a standard form p defined a priori (e.g., a standard Gaussian distribution) which prevents learning degenerate distributions. Also, crucially for our goals, this loss can be *exploited* to model different shape regularization constraints (see next Section 4.2.2). Our complete loss function then reads as

$$\begin{aligned} \mathcal{L}_{\text{ShaRP}}(\mathbf{X}, \hat{\mathbf{X}}, \bar{\mathbf{Y}}, \hat{\mathbf{Y}}, \Theta) &= \mathcal{L}_{\text{recon}}(\mathbf{X}, \hat{\mathbf{X}}) + \rho \mathcal{L}_{\text{class}}(\bar{\mathbf{Y}}, \hat{\mathbf{Y}}) + \beta \mathcal{L}_{\text{reg}}(\Theta) \\ &= \mathcal{L}_{\text{SSNP}}(\mathbf{X}, \hat{\mathbf{X}}, \bar{\mathbf{Y}}, \hat{\mathbf{Y}}) + \beta \mathcal{L}_{\text{reg}}(\Theta), \end{aligned} \quad (4.5)$$

where we make the connection to the SSNP loss explicit.

Table 3: Overview of desirable properties in three strongly related techniques (VAE, SSNP, ShaRP), together with whether each technique fulfills the property completely (●), partially (◦), or does not fulfill it (×).

Property	VAE	SSNP	ShaRP (ours)
Projection-oriented	×	●	●
Can use (pseudo-)labels	×	●	●
DBM generation	×	●	●
Latent space regularization	●	×	●
Shape regularization	◦	×	●
Non-Euclidean projection	◦	×	●

By using a suitable *sampling* process, the clusters emerging in the projection will be shape-regularized. For example, a 2D Gaussian sampling distribution yields *elliptical* shapes (see Fig. 4.2c) because the equidensity contours of a 2D Gaussian are ellipses. This is dependent on $\mathcal{L}_{\text{reg}}$ preventing the degenerate learning of low (respectively, high) variances, which would give rise to point-like (resp., line-like) shapes in the projection.

Table 3 summarizes the differences between using a pure VAE, SSNP, and ShaRP. A VAE is good at producing a latent representation that enables downstream tasks, so its latent space usually has more than 2 or 3 dimensions and visualizing (projecting) it is not a main concern. This also means that, even though VAEs are *in theory* capable of shape regularization and non-Euclidean projection, they meet such goals only partial due to the aforementioned reason. SSNP is designed to generate projections, and has support for (self-)supervision through (pseudo-)labels. It performs no regularization on the learned latent space due to the use of a vanilla auto-encoder, which also means it is unable of shape regularization or non-Euclidean projections. ShaRP fulfills all of the listed properties. It generates projections and DBMs on a regularized latent space, allows for (self-)supervision, and has the extra capabilities of shape regularization and non-Euclidean projection.

4.2.2 Controlling cluster shapes

We use as regularization targets the following shapes.

Ellipses. Consider a diagonal Multivariate Normal distribution, *i.e.*, $\mathbf{z}_i \sim \mathcal{N}(\boldsymbol{\mu}, \text{diag}(\sigma^2))$. The natural prior to use here is the standard Multivariate Normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$ which simplifies sampling, propagating gradients, and calculating the KLD loss (Kingma and Welling, 2014). By using this prior, we encourage learned probability distributions to be as close as

possible to a standard Gaussian. Hence, the learned projection will output data clusters that resemble circles or ellipses (see next Fig. 4.8).

Using a Gaussian sampling distribution is standard for VAEs. For our projection goals, tweaking the sampling distribution and using suitable priors allows favoring different cluster shapes. We can use *any* distribution as long as we have (i) access to log-probabilities of samples under the learned distribution and the prior; (ii) a way to propagate gradients through the sampling process (using a reparametrization trick or otherwise).

Access to the log-probabilities of samples under learned distributions and the prior removes the (constraining) need to analytically calculate the KLD since we can re-express it as a sample-based computation as

$$D_{\text{KL}}(q_{\theta}||p) \approx \frac{1}{m} \sum_{i=1}^m (\log q_{\theta}(\mathbf{z}_i) - \log p(\mathbf{z}_i)), \quad (4.6)$$

where the approximation holds if $\mathbf{z}_i \sim q_{\theta}(\cdot)$. Our next examples of regularization shape targets use Eqn. 4.6 for computing the (approximate) KLD.

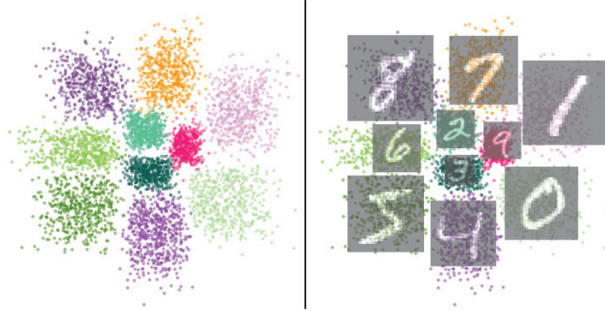


Figure 4.3: Rectangular shaping can be convenient for data labeling tasks, as illustrated by the right image where class representatives are overlaid atop their clusters. We achieve this using a Generalized Normal distribution, here shown on the MNIST dataset for $\omega = 10$ (left).

Rectangles. To create rectangular clusters, we use a Generalized Normal ($\mathcal{GN}$) probability distribution. It introduces an additional shape parameter to the Gaussian (here denoted ω) and has a density function of the form

$$p(x|\mu, \alpha, \omega) \propto \exp(-(|x - \mu|/\alpha)^\omega).$$

Tuning ω makes the distribution tails heavier or lighter. This is akin to the Minkowski p-norm where $p \rightarrow \infty$ values (analogous to ω) make sets of equidistant points approach axis-aligned squares instead of circles ($p = 2$). Using this distribution for sampling, with a high ω , yields cluster shapes that resemble squares/rectangles instead of ellipses (see Figs. 4.3, 4.4).

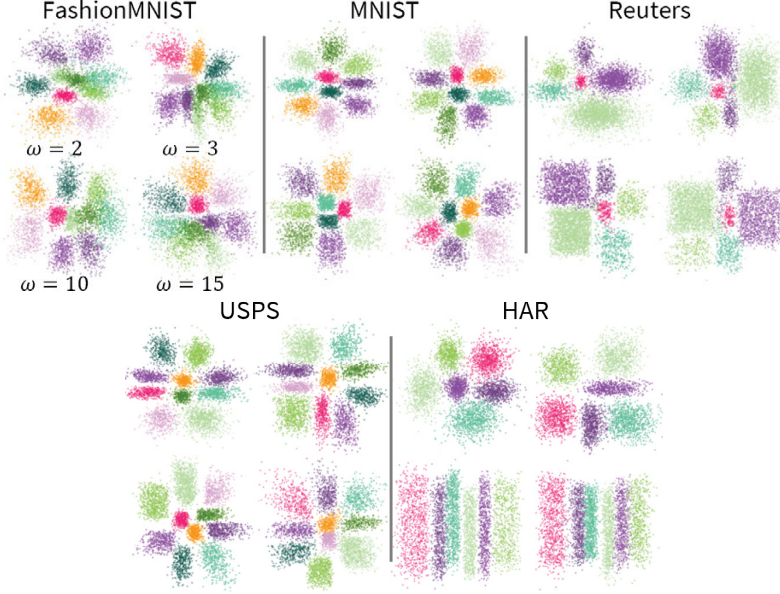


Figure 4.4: Demonstration of shape regularization towards rectangles for different values of ω across five datasets. The strength of the “squarification” effect consistently increases with increasing ω , as expected.

Triangles. We first define an equilateral triangle in $\mathbb{R}^2$ by arranging its vertices $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ in a matrix as

$$\mathbf{T} = [\mathbf{v}_1 \ \mathbf{v}_2 \ \mathbf{v}_3] = \begin{bmatrix} 0 & 1/2 & 1 \\ 0 & \sqrt{3}/2 & 0 \end{bmatrix}.$$

To remove bias in the base shape, we use a base equilateral, unit-edge, triangle. The initial bias can be overcome through the training process via the following necessary extension that we add to the sampling scheme. Any convex combination of the barycentric coordinates $\mathbf{v}_1, \mathbf{v}_2$, and $\mathbf{v}_3$ gives a point inside the triangle. Formally: for a vector $\mathbf{w} = [w_1 \ w_2 \ w_3]^T$ with $w_i \in [0, 1]$, $1 \leq i \leq 3$ and $\sum_{i=1}^3 w_i = 1$, $\mathbf{p} = \mathbf{T}\mathbf{w}$ is inside the triangle.

We now need to use a sampling distribution in ShaRP that generates vectors with the same properties as $\mathbf{w}$ above. The Dirichlet probability distribution

$$\mathbf{w} \sim \text{Dir}(\alpha_1, \alpha_2, \alpha_3) \Rightarrow \mathbf{w} \in [0, 1]^3, \quad \sum_{i=1}^3 w_i = 1 \quad (\alpha_i > 0, \forall i)$$

does exactly that. We choose as prior the “uniform” distribution on the triangle, which corresponds to $\text{Dir}(1, 1, 1)$.

If we stopped here, our algorithm would fail to learn a *useful* embedding since every data point will draw samples from the same base triangle, *i.e.*,

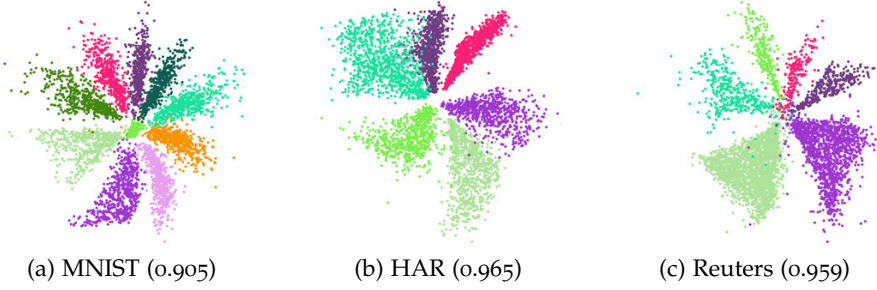


Figure 4.5: The results of our Triangular shaping sampling scheme over 3 different datasets. DSC values (in brackets) are close to the best value possible, indicating good class separability.

the encoding layer will map all points to the same region in 2D space. Hence, we augment our sampling scheme to allow triangles to be rotated, scaled, and translated.

The set of learned parameters used to force shapes into triangles is $\theta = (\phi \in [-\pi, \pi], s_x \in \mathbb{R}_+, s_y \in \mathbb{R}_+, t_x \in \mathbb{R}, t_y \in \mathbb{R}, \alpha_1, \alpha_2, \alpha_3)$. Here, ϕ is a rotation angle; s_x and s_y are scaling factors of the x and y axes; t_x and t_y are translation amounts in the x and y directions; and α_i are the sampling distribution parameters. A forward pass through this layer is given by

$$\mathbf{w} \sim \text{Dir}(\alpha_1, \alpha_2, \alpha_3)$$

$$\mathbf{p} = \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \mathbf{T}\mathbf{w} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

As a result, we get triangle-shaped clusters. This parameterization can generate every possible triangle in $\mathbb{R}^2$ and is more convenient than learning triangle vertices directly. We can even add individual regularization losses depending on the parameter’s semantics. For example, we choose *not* to add regularization to ϕ , to allow it to freely range over its domain; we regularize s_x, s_y towards 1 and t_x, t_y towards 0. Figure 4.6 shows the effect this has on different datasets, and also what happens when we remove a degree of freedom from the sampling scheme, namely setting the t_x, t_y translation amounts to 0.

Convex polygons. The triangle sampling scheme described above lends itself easily to generalization to *any* convex polygon in $\mathbb{R}^2$. This can be achieved by observing that a v -dimensional Dirichlet distribution can be used to generate barycentric coordinates $\mathbf{w} \sim \text{Dir}(\alpha_1, \dots, \alpha_v)$ of a v -sided polygon. In practice, sampling in triangular shapes can be viewed as a special case where the convex polygon of interest has $v = 3$.

We build a matrix $\mathbf{V} \in \mathbb{R}^{2 \times v}$ of the base convex polygon’s v vertices in $\mathbb{R}^2$ with equally spaced points around a circle of radius 1 for the same

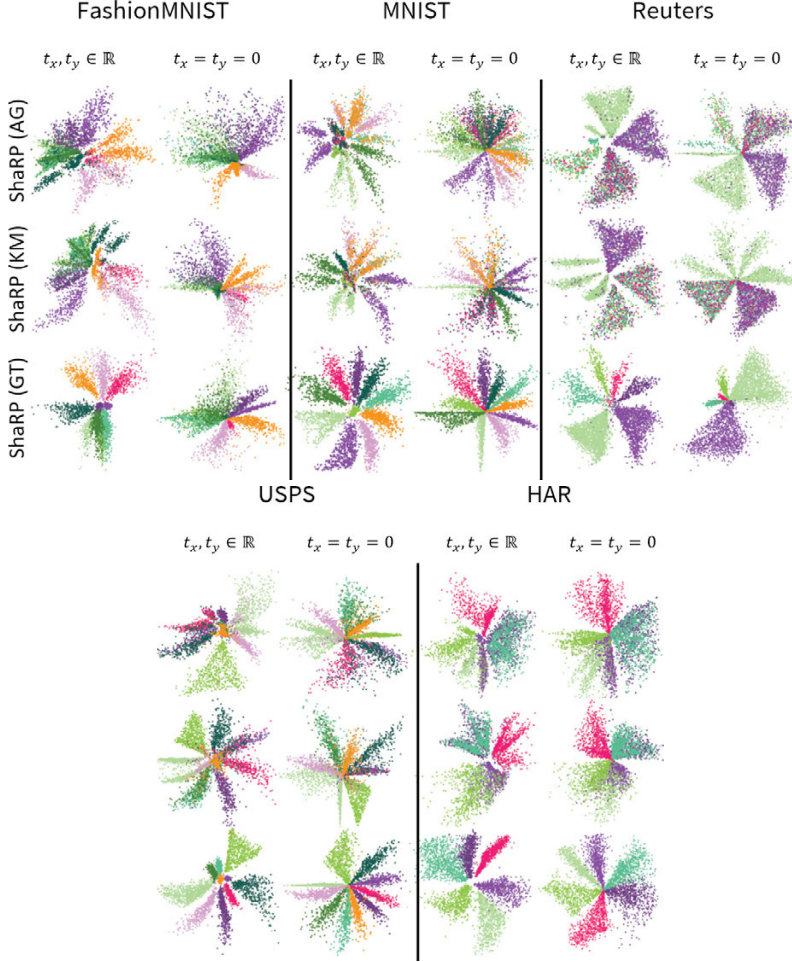


Figure 4.6: Triangle-oriented shape regularization demonstrated for five of the studied datasets. We present all three variants of ShaRP and also explore the impact of (dis)allowing the translation of triangles in space. We notice that ShaRP is always able to find a sensible projection where distinct class clusters can be identified, even when we take away some degrees of freedom of the technique (e.g., forcing $t_x = t_y = 0$)

reasons as in the triangular case. The bias caused by this choice of initial coordinates is, as earlier, overcome by augmentations that allow for rotation, scaling, and translation. We also see that any point $\mathbf{p} = \mathbf{V}\mathbf{w}$ is inside the specified polygon. To achieve shape regularization, the prior distribution should encourage sampling to occur in all regions of the polygon, which is achieved when all the α_i are set to 1.

4.2.3 Projection on non-Euclidean spaces

Projection techniques most often concern themselves with finding “compressed” representations of high-dimensional data in a lower dimensional space $\mathbb{R}^q$ that has a standard *Euclidean* structure. This seemingly natural design choice leads to projections where one corner of the projected space is at the largest distance possible from its diagonally opposing corner — assuming, without loss of generalization, that projected points range in $[0, 1]^q$. This space cannot represent well certain similarity relationships between categories. For example, consider a dataset having K classes, where each class i , $1 \leq i \leq K$, is similar to class $i + 1 \bmod K$. A projection of this dataset in Euclidean space would have to distort *at least one* of these similarity relationships. Well-known instances hereof are projections of the 3D earth surface (with geodesic distances between countries being the similarity relationships) by classical 2D cartographic methods (van Wijk, 2008).

We address this problem as follows. The sampling mechanism of ShaRP *not only* enables shape regularization, but also allows for the possibility of projecting onto spaces *other than* $\mathbb{R}^2$. For example, using a 3D von Mises-Fisher distribution (vMF) results in projecting points in $\mathcal{S}^2$, the 2-dimensional sphere — i.e., the boundary of a 3D ball. Previous work has explored the use of such distributions in VAEs (Davidson et al., 2018), and our work is the first, to the best of our knowledge, to use this in a specific architecture (see Fig. 4.1) with the primary goal of producing high quality data projections in non-Euclidean space.

Figure 4.7 (top) shows this for a subset of the ‘Quick, draw!’ dataset (Ha and Eck, 2018). From all the 345 classes of the dataset, we selected the following ten to explore, due to similarities in their appearances: Ambulance, Bicycle, Bulldozer, Car, Firetruck, Motorbike, Pickup truck, Police car, Truck, and Van. We expect good projections to capture true data similarities, *e.g.*, show ambulances next to vans, bicycles next to motorbikes, and so on.

Images (a-c) show three viewpoints of the spherical projection created by ShaRP, with points slightly shaded to represent the spherical surface. In (a) we see the classes Car (bottom-right of sphere) and Police Car (top), which we expect to be similar and therefore projected in neighboring regions; in (b) we see to the right the classes Bicycle and Motorbike — which are again similar and projected next to each other; in (c) we see the Pickup truck (top left), Truck (center), and Firetruck (bottom) are projected in adjacent regions as well.

Figure 4.7 (top) d-f show three viewpoints of the same dataset, projected in 3D *Euclidean* space using t-SNE. These views show some similarity of the same-class samples, visible as close points having the same color. Yet, the delineation of same-class points as *clearly separated* same-color regions in the projection, visible with the ShaRP spherical projection (a-e), is now lost.

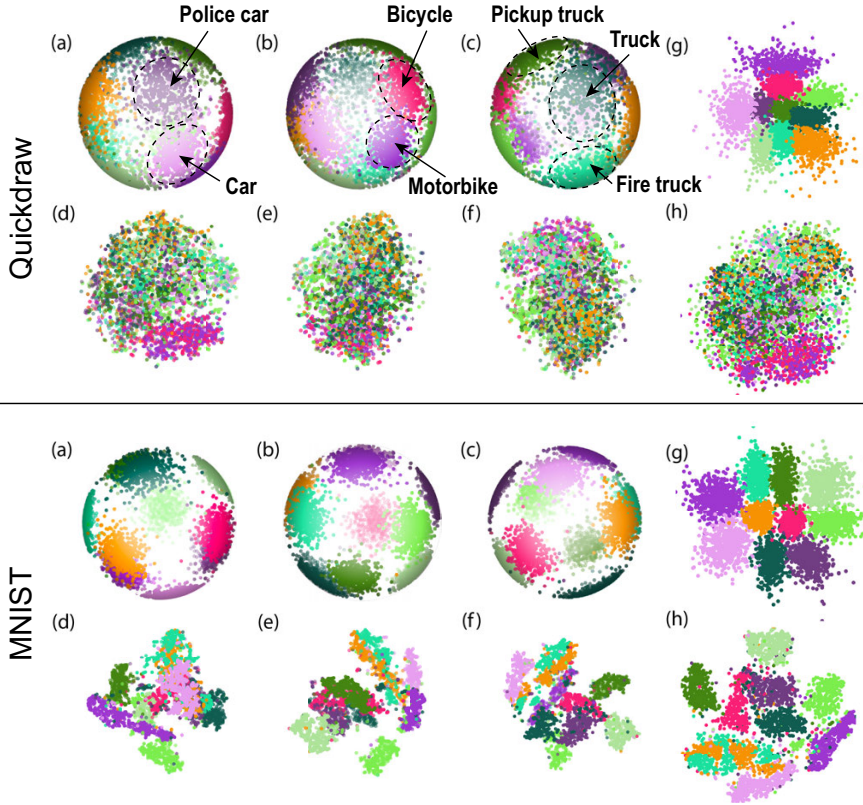


Figure 4.7: Projections of the Quickdraw (top) and MNIST (bottom) datasets. Using ShaRP’s spherical sampling scheme plots points on the 2D surface of a 3D sphere (different viewpoints shown in a-c). Using such a manifold allows for wrap-around behavior that allows showing more complex similarity relationships of data points. This is in stark contrast with a 3D t-SNE plot (three viewpoints of which are shown in d-f) where similar groups are not easily identifiable. In 2D, a standard t-SNE plot (h) does not make the visualization better; and standard 2D ShaRP (g) can only show groupings in a limited way.

The key problem for t-SNE is that it uses the *entire* 3D space to project data. Given the earlier-mentioned tendencies of t-SNE to create round, organic-like shapes, and also to fill in the projection space with these shapes, this yields a sphere filled in with the projected points. Finding out how such points group into same-class regions, representing class-consistent groupings, is extremely hard, even when examining this 3D scatterplot from multiple viewpoints, due to occlusion and depth effects. Finding out how same-color point groups neighbor each other is even harder for the same reason. The above are fundamental challenges of exploring 3D scatterplots

created by projections, *i.e.*, not limited to t-SNE (Tian et al., 2021). Additional interactive exploration tools can help finding better viewpoints but cannot eliminate the occlusion and inherent difficulties of understanding a dense 3D scatterplot (Castelein et al., 2023). In contrast, our spherical space considerably decreases the occlusion problems (maximally two points can be on the same view line). Also, as explained earlier, the spherical space allows for a more flexible point placement to model similarities than Euclidean (3D) space.

One can now ask whether simply avoiding 3D projections and using 2D ones would not solve the above issues. Figure 4.7 (top) g,h shows the projections of the same dataset to 2D using ShaRP, respectively t-SNE. We see that ShaRP renders the similarity of same-class points far better than t-SNE due to its shape constraints. We also see an additional advantage of the earlier spherical space: whereas the spherical projection shows that the green and pink classes are similar (since they are close on the sphere, see images (a) and (b)), the 2D projection places these far apart from each other, due to its need to render the other similarities of the respective classes with their neighbors. Clearly, assessing 3D projections — or, more generally, 3D scatterplots — is hard to do in a static 2D view as shown in this chapter. For such plots, interaction is paramount, as explained in e.g. (Poco et al., 2011; Coimbra et al., 2016; Tian et al., 2021; Castelein et al., 2023). Our work is openly available for users to try out such interactive exploration (see GitHub link listed earlier in this section).

Figure 4.7 (bottom) shows the same experiment as above, now done for the MNIST dataset. While we see a clearer separation of the same-color clusters in the t-SNE view, this separation is massively better with spherical ShaRP. The same quality difference is visible in 2D ShaRP *vs* 2D t-SNE.

It is important to note that, for the evaluation of such non-Euclidean projections using quality metrics, we adapt distance calculations to use the appropriate geodesic distance which, on a unit sphere (S^2), is the great-circle distance

$$d_{S^2}(\mathbf{x}, \mathbf{y}) = 2 \mid \arcsin(0.5 \|\mathbf{x} - \mathbf{y}\|) \mid \in [0, \pi].$$

Table 4 summarizes our proposed mechanisms for controlling cluster shapes by sampling distributions in both 2D Euclidean and spherical S^2 spaces.

4.3 EVALUATION

We next discuss how ShaRP gives direct control over cluster shapes while learning to project data (Fig. 4.3.1), the quality of ShaRP projections (Section 4.3.2), and how tuning a single parameter controls the shape regularization strength (Section 4.3.3). We also show how ShaRP can be used

Table 4: Mechanisms offered by ShaRP to control the cluster shapes and/or the projection space.

Sampling	Prior	Shape	Space
$\mathbf{z} \sim \mathcal{N}(\mu, \text{diag}(\sigma^2))$	$\mathcal{N}(0, I)$	$\bigcirc$	$\mathbb{R}^q$
$\mathbf{z} \sim \mathcal{GN}(\mu, \alpha, \omega)$	$\mathcal{GN}(0, 1, \omega)$	$\square$	$\mathbb{R}^q$
$\mathbf{z} \sim \text{Dir}(\alpha_1, \alpha_2, \alpha_3)$			
$\mapsto \begin{bmatrix} s_x \cos \phi & -s_y \sin \phi \\ s_x \sin \phi & s_y \cos \phi \end{bmatrix} \mathbf{V} \mathbf{z} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$	$\text{Dir}(1, 1, 1)$	$\triangle$	$\mathbb{R}^2$
$\mathbf{z} \sim \text{vMF}(\mu, \kappa)$	$\text{vMF}(\mu, 0)$	$\bigcirc$	$\mathcal{S}^2$

to compute decision maps for exploring trained classification models and compare it with other recent methods for the same task (Section 4.3.5). Finally, we discuss ShaRP’s computational scalability (Section 4.3.6).

Datasets. We use 15 datasets for evaluation (Table 5) with different classification difficulties, dimensionality, data types (images, motion data, text), all often used in DR evaluations (Espadoto et al., 2021b).

Techniques. We compare ShaRP with t-SNE, UMAP, and Isomap, due to their wide adoption in the DR arena. We also compare with Auto-Encoders since they are a key building block of our technique; with SSNP since we are extending it; and with NNP (Espadoto et al., 2019a), a technique that learns to imitate projections, here trained to imitate t-SNE. For both ShaRP and SSNP, we use three different label sources: from the ground truth of the dataset (GT); and pseudolabels created by the K-Means (KM) (Lloyd, 1982) and Agglomerative (AG) (Kaufman and Rousseeuw, 1990) clustering techniques.

Hyperparameter settings. We train ShaRP with the Adam optimizer with default parameter settings. We add L_2 regularization to the network’s bottleneck layer with a coefficient of 0.5. We use $\rho = 1$ and $\beta = 0.1$ and mini-batches of 256 data points.

4.3.1 Generating shape-regularized projections

Figures 4.8-4.16 show how ShaRP regularizes projections for our 15 studied datasets and compares these results with eight other projection techniques. All images were generated using the same hyperparameter values, showing the robustness of ShaRP to different datasets. Instead of producing scatterplots where cluster shapes, sizes, and intercluster spacing are widely different (as with t-SNE and UMAP), ShaRP generates a more similar representation of the high-dimensional data in each 2D projection (intra-projection regularization). Also, the visual signature obtained is consistent

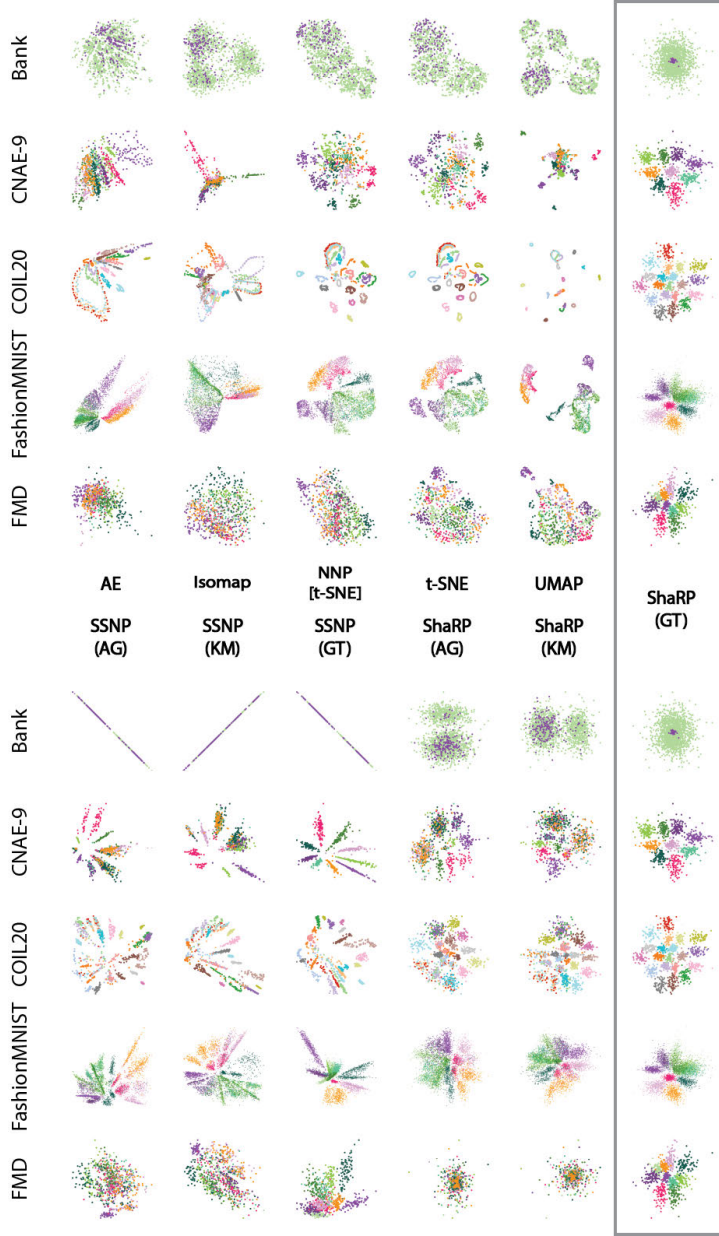


Figure 4.8: Projections of 15 datasets by 10 projection techniques (continued in Figs. 4.15, 4.16, see end of chapter). Our ShaRP method produces cluster shapes regularized towards a user-chosen target — here, ellipses — and can handle diverse data distributions, both when using (GT) labels or pseudolabels generated by K-Means (KM) or agglomerative clustering (AG).

Table 5: Datasets used in our evaluation.

Dataset	Dimensionality (n)	# classes (K)
Bank (Moro et al., 2014)	63	2
CNAE-9 (Ciarelli and Oliveira, 2009)	856	9
COIL20 (Nene et al., 1996)	400	20
FashionMNIST (Xiao et al., 2017)	784	10
FMD (Sharan et al., 2010)	1536	10
HAR (Anguita et al., 2012)	561	6
Hate Speech (Davidson et al., 2017)	100	3
IMDB (Maas et al., 2011)	700	2
MNIST (LeCun et al., 1998)	784	10
Quick, draw! (Ha and Eck, 2018)	784	10
Reuters (Thoma, 2017)	5000	6
Sentiment (Kotzias et al., 2015)	200	2
Spambase (Hopkins et al., 1999)	57	2
SVHN (Netzer et al., 2011)	1024	10
USPS (Hull, 1994)	256	10

throughout datasets (inter-projection regularization). The learned projections do well with respect to quality metrics (see Fig. 4.9 and its discussion in Section 4.3.2).

This comparison shows how the choice of projection algorithm has a strong impact on the structures that arise in the projection plots. Worthy of note are:

For the COIL20 dataset (Fig. 4.8), all of Isomap, t-SNE, and UMAP produce ring-like structures (more deformed in Isomap) with different degrees of spread and intercluster spacing (most visible in UMAP, with low spread and big spacing between clusters). These seem to point to pairwise sample similarities forming a chain, with points on opposite sides of the rings being most different from each other. In contrast, ShaRP is able to effectively project all samples without wasting the space necessary to “draw” a ring; it draws filled circles with the data points instead, and arranges them in 2D space with approximately uniform intercluster spacing. One may wonder who is ‘right’ here. We point out that PCA, a well-known linear projection method that should have no trouble preserving such (near-)2D topology, does not produce such structures in 2D. Also, changing the t-SNE perplexity parameters can make these rings disappear — see *e.g.* the related analysis of how setting perplexity can be tricky for t-SNE (Wattenberg et al., 2016). As such, we conjecture that the existence of such rings in data space

is not clear — it is, at the very least, contingent on the representation used. Our method, by design, does not create these rings because of its shape regularization. This behavior is undoubtedly destructive, in the sense that if a dataset *does* possess such clear patterns in the data space, ShaRP *will* erase them and favor whichever shape regularization target is used. Whether this is desirable is left for the user to decide. Note also that our approach comes at the cost of introducing some false neighbors — which leads to a lower trustworthiness value — while producing better DSC and Stress values.

Datasets with only two classes (e.g., Bank, IMDB, SpamBase) seem to draw out problematic behavior in SSNP. Namely, as we also discuss next in Section 4.3.2, the absence of dedicated neurons in the output layer for both positive and negative class *collapses* the projections of different-class data points into a single line. This causes values of Stress that reach 6 orders of magnitude above what is produced by other techniques. ShaRP does not suffer of this problem, being able to reliably place points with different (pseudo-)labels in distinct regions of space.

Finally, for the Hate Speech dataset, all studied techniques have difficulty coming up with an easily visually separable projection. However, the patterns present in different projections are qualitatively distinguishable: for Isomap, we see lots of intermixing around the middle of the projection, but not toward the edges; in t-SNE, we see small groups of same-class samples, mixed with each other; UMAP seems to collapse most of the samples towards the same spatial region. ShaRP is able to create some “pure” clusters even in absence of ground truth labels (see ShaRP(KM)). If ground truth labels are provided, the two classes become even more distinguishable, even if they’re projected concentrically.

Across all datasets, we consistently see ShaRP’s ability to regularize cluster shapes towards a desired form, without the need to change hyperparameters to adapt to new datasets.

4.3.2 Measuring projection quality

We evaluate ShaRP by trustworthiness, continuity, Shepard correlation, normalized stress, neighborhood hit, and distance consistency (see Sec. 2.4.2). We compute these metrics over all datasets using a Gaussian sampling layer which produces ellipse-like clusters. Figure 4.9 shows the metric values over all datasets for ShaRP and the other six evaluated projection techniques. Since Stress (Q_S) has values in an unbounded range, we chose not to visualize it here along the other metrics. We report Q_S values, together with detailed numerical values for all other metrics, in Appendix A.2.

For the Q_S metric, ShaRP avoids very high values (present in t-SNE, *all* studied datasets; UMAP and AEs, some datasets). We do, however, have higher Stress than SSNP, since we *force* clusters into desired shapes, which

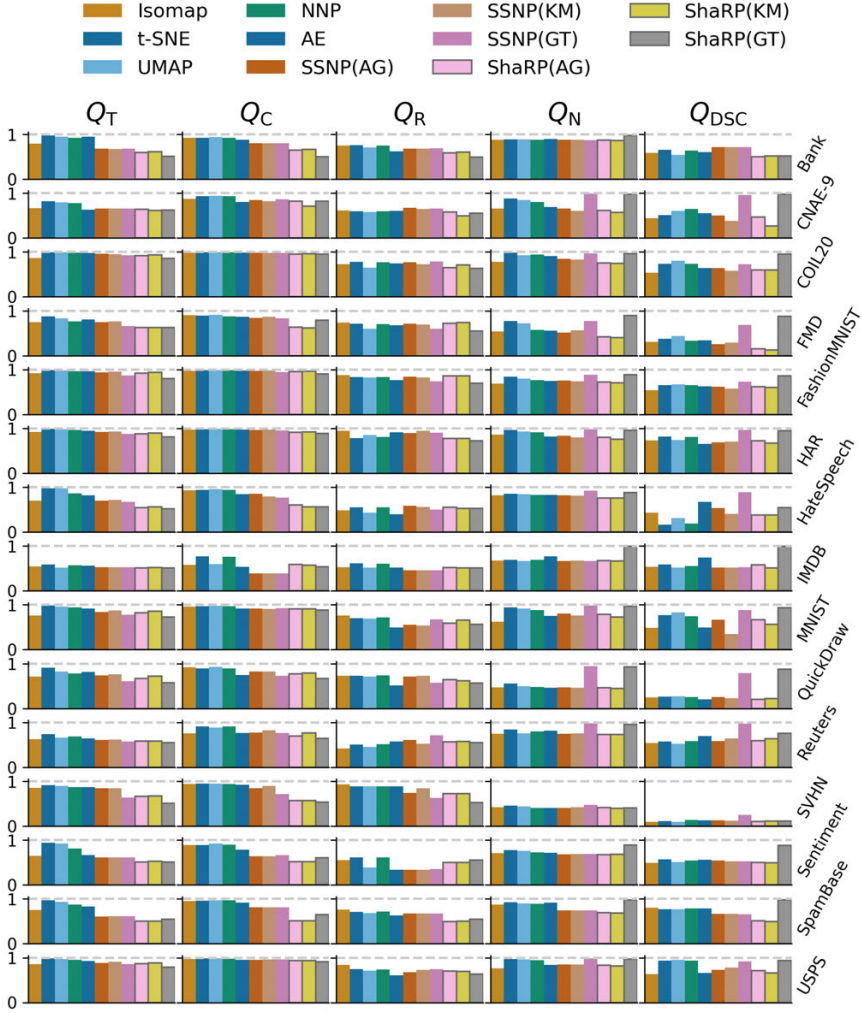


Figure 4.9: Performance of studied projection algorithms according to standard quality metrics: Trustworthiness Q_T , Continuity Q_C , Shepard Goodness Q_R , Neighborhood Hit Q_N , and Distance Consistency Q_{DSC} (higher is better, see Sec. 2.4.2). We observe a loss in terms of quality metrics due to shape regularization. Q_{DSC} is also highest for the (GT) variants of SSNP and ShaRP, which is expected as *true* class information is being used. For detailed measurements including the Stress metric, see Appendix A.2.

Table 6: Non-aggregated quality metrics for different sampling schemes within ShaRP. Rightmost column shows mean square error (MSE) for training.

Dataset	Shape	Q_T	Q_C	Q_R	Q_S	Q_N	Q_{DSC}	MSE
Fashion	$\bigcirc$	0.835	0.907	0.463	0.880	0.829	0.766	0.048
	$\square[\omega = 5]$	0.825	0.915	0.485	0.892	0.842	0.816	0.050
	$\square[\omega = 15]$	0.829	0.916	0.495	0.891	0.813	0.771	0.049
	$\triangle$	0.826	0.898	0.482	0.875	0.895	0.781	0.050
	$\triangle^\dagger$	0.807	0.808	0.208	0.866	0.834	0.641	0.053
	$\mathcal{S}^2$	0.815	0.910	0.308	0.751	0.952	0.951	0.049
HAR	$\bigcirc$	0.823	0.810	0.521	0.749	0.965	0.829	0.010
	$\square[\omega = 5]$	0.825	0.880	0.582	0.749	0.952	0.936	0.010
	$\square[\omega = 15]$	0.829	0.886	0.513	0.768	0.928	0.909	0.010
	$\triangle$	0.828	0.854	0.563	0.662	0.978	0.921	0.010
	$\triangle^\dagger$	0.830	0.824	0.426	0.717	0.980	0.928	0.010
	$\mathcal{S}^2$	0.823	0.898	0.423	0.522	0.990	0.990	0.009
MNIST	$\bigcirc$	0.732	0.897	0.251	0.859	0.977	0.962	0.055
	$\square[\omega = 5]$	0.735	0.896	0.271	0.878	0.979	0.961	0.055
	$\square[\omega = 15]$	0.735	0.900	0.230	0.864	0.968	0.948	0.055
	$\triangle$	0.747	0.874	0.119	0.850	0.986	0.896	0.054
	$\triangle^\dagger$	0.738	0.799	0.148	0.844	0.959	0.769	0.056
	$\mathcal{S}^2$	0.735	0.907	0.219	0.722	0.993	0.994	0.053
Reuters	$\bigcirc$	0.554	0.701	0.104	0.676	0.962	0.851	0.001
	$\square[\omega = 5]$	0.556	0.699	0.300	0.714	0.975	0.933	0.001
	$\square[\omega = 15]$	0.558	0.715	0.256	0.760	0.977	0.953	0.001
	$\triangle$	0.560	0.696	0.265	0.581	0.981	0.953	0.001
	$\triangle^\dagger$	0.561	0.637	0.002	0.665	0.977	0.793	0.001
	$\mathcal{S}^2$	0.555	0.709	0.283	0.411	0.981	0.971	0.001
USPS	$\bigcirc$	0.802	0.922	0.285	0.774	0.973	0.931	0.044
	$\square[\omega = 5]$	0.799	0.918	0.350	0.793	0.975	0.948	0.045
	$\square[\omega = 15]$	0.798	0.919	0.329	0.792	0.962	0.934	0.045
	$\triangle$	0.823	0.898	0.363	0.775	0.992	0.887	0.043
	$\triangle^\dagger$	0.804	0.826	0.185	0.763	0.972	0.837	0.045
	$\mathcal{S}^2$	0.826	0.935	0.296	0.559	0.993	0.993	0.040
Shape		Sampling Scheme						
$\bigcirc$		ellipses, Gaussian sampling						
$\square[\omega = k]$		squares, generalized Normal sampling with $\omega = k$						
$\triangle$		triangles, Dirichlet sampling						
$\triangle^\dagger$		triangles, Dirichlet sampling with $t_x = t_y = 0$						
$\mathcal{S}^2$		spherical, von Mises-Fisher sampling						

can require projected (2D) distances to be quite different from data-space distances. As explained in the opening of this chapter, if, for example, data is spread into Gaussian clusters, and we use ShaRP with the elliptical cluster constraint, then we expect to see good quality metric values (Stress and others). Conversely, if the data is spread in different ways than the visual signature selected by the user for ShaRP, then some quality metrics may decrease for ShaRP. Given that our Stress is still lower than t-SNE, UMAP, and AE, we believe this is a reasonable trade-off. For the same reason, Q_T and Q_C tend to have (even if slightly) lower values in ShaRP than in reference projection algorithms — since shaping can introduce (resp. miss) data space neighbors. Overall, we claim that ShaRP offers its capability of shape regularization with slight impacts to quality metrics — stronger for high values of β , see Eq. (4.5). It is worth noting that, for their AG and KM versions, both SSNP and ShaRP can be held back by the clustering algorithm’s ability to properly group the dataset into classes.

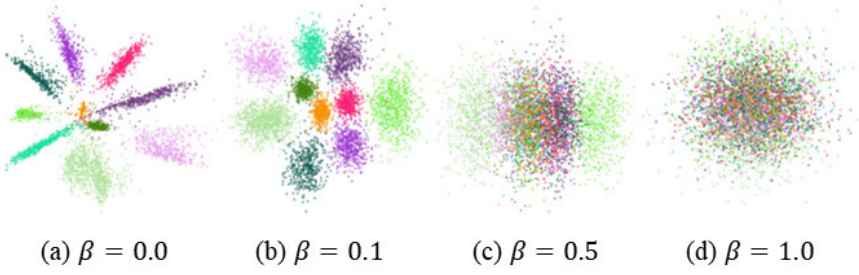
To test how ShaRP’s support of different regularization shapes affects projection quality, we ran ShaRP to produce clusters in five shapes — ellipses (using Gaussian sampling); rectangles ($\omega = 5$ and $\omega = 15$, see Section 4.2.2); and triangles (translated in projected space and respectively forced to $t_x = t_y = 0$), for all 5 tested datasets. Table 6 shows the quality metrics for these settings, for five of the tested datasets (for brevity; for the other 10 datasets, the values are similar). We see little variation in these metrics. This points to the robustness of ShaRP and further supports our claim that controlling the visual signatures of projections can be done without (strongly) influencing quality metric values.

4.3.3 Control of shape regularization intensity

We adjust the amount of shape regularization through the β multiplier in the loss function (Eq. (4.5)). Figure 4.10 shows this: larger β values force clusters to conform to the shape generated in the sampling layer — ellipses, in this case. Exaggerated shape regularization (high β), however, makes ShaRP favor ‘shape over data’ too much and creates projections which cannot properly depict data — sampling from a distribution similar to the prior overshadows producing a sensible embedding. The table below in Fig. 4.10 gives the quality metrics for the different β settings. In our tests, we have found a value of $\beta = 0.1$ to give consistently good results.

4.3.4 Faithfulness to data class structure

As ShaRP is a method that uses (pseudo-)labels $\tilde{Y}$, it is important to assess whether the creation of clusters in a projection is artificial, and therefore less trustworthy, similar to earlier studies that aim to assess class faithful-



	$\beta =$	0.0	0.05	0.1	0.25	0.5	1.0
Trustworthiness		0.87	0.81	0.79	0.77	0.66	0.54
Continuity		0.95	0.93	0.85	0.74	0.70	0.56
Shepard Corr.		0.39	0.30	0.27	0.26	0.21	0.09
Stress		0.93	0.84	0.78	0.66	0.57	0.54
Neighborhood Hit		0.99	0.99	0.97	0.91	0.60	0.45

Figure 4.10: The β coefficient (Eq. (4.5)) controls the shape regularization strength, shown here on the USPS dataset. $\beta = 0$ approximately reproduces SSNP (a). Increasing it (b, c) progressively forces the learned clusters into circular shapes, up to the point where the projection is of low quality (d).

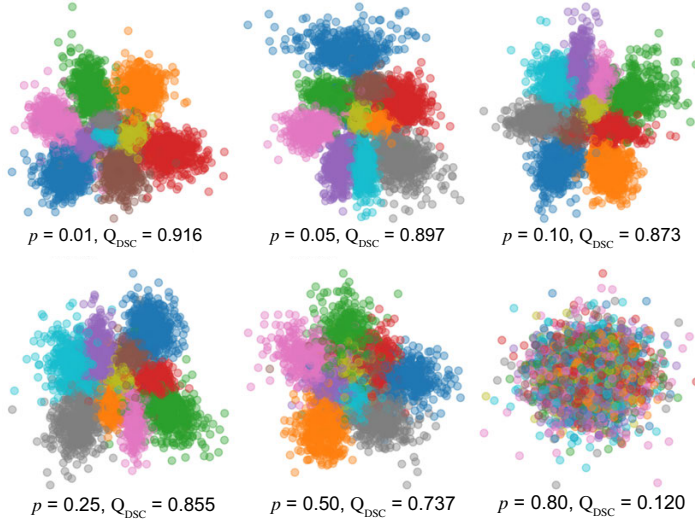


Figure 4.11: Progressively mixing classes in the MNIST dataset (*i.e.*, increasing the mixing fraction p) leads to the progressive degradation of the projection produced by ShaRP, as visible above and also measured by the class separability metric Q_{DSC} . Even though ShaRP has access to the (noisy) labels, they do not artificially create separated clusters — as is expected from projection methods.

ness in projections (Colange et al., 2020; Jeon et al., 2024a). To this end, we conduct an experiment in which we randomly assign a different label $y'_i \in \{1, \dots, K\}$ to increasing fractions $p \in [0, 1]$ of the samples of each class in a dataset — we choose MNIST due to expected reader familiarity. This *class mixing* should produce correspondingly *mixed* projections as we increase p . Figure 4.11 shows exactly this effect: As we increase p , the observed visual separation of same-label clusters decreases; also, the class separability metric Q_{DSC} *monotonically* decreases.

This experiment provides evidence that ShaRP does not artificially create clusters based on the (pseudo-)label information it is supplied. That is to say, ShaRP’s shape regularization effect still preserves intra-class similarities, and therefore data class structure, as we would expect from a sensible DR algorithm.

4.3.5 Generating decision boundary maps

Visualizing how a classifier partitions the input space into regions corresponding to different classes can give insight into its correctness and robustness. For high-dimensional inputs, this visualization uses a projected space through so-called Decision Boundary Maps (see Sec. 2.6). A key factor to the quality of DBMs is the inverse projection used. We next focus on comparing with SDBM given that, from all existing techniques in this area — that is, Standard DBM (Rodrigues et al., 2018), SDBM (Oliveira et al., 2022), and DeepView (Schulz et al., 2015) — this technique strikes the best balance of quality and computational scalability (Wang et al., 2023).

Figure 4.12 compares SDBM and ShaRP for visualizing the decision boundaries of a given, (per-dataset) fixed, classifier. We chose as classifier model a shallow Multilayer Perceptron — 3 layers with sizes 512, 128, 32 — with ReLU activations. We additionally encode the confidence of the classifier — *i.e.*, the probability assigned to the most probable class — in the saturation of the DBM colors, giving extra insights into the classification quality per region. Note that, since ShaRP learns to jointly project *and* classify, we could use the classifier head of its decoder over a grid of points in the latent space, immediately obtaining information on which class is projected to which region, without any additional training. Also, ShaRP outputs a *probability distribution* over classes, enabling the visualization of uncertainty — *e.g.*, entropy — in the DBMs, without any additional effort. However, as mentioned above, we choose not to use ShaRP’s built-in classifier, but an externally given one. This decouples how the DBM is generated from what it is used to visualize — which, we argue, makes full sense. Indeed, users likely want to use a DBM visualization to inspect their own classifiers, and not the one in ShaRP.

In Fig. 4.12, we see that SDBM seems to fail to generate sensible DBMs for cases where there are only two classes in the dataset (see *e.g.* datasets

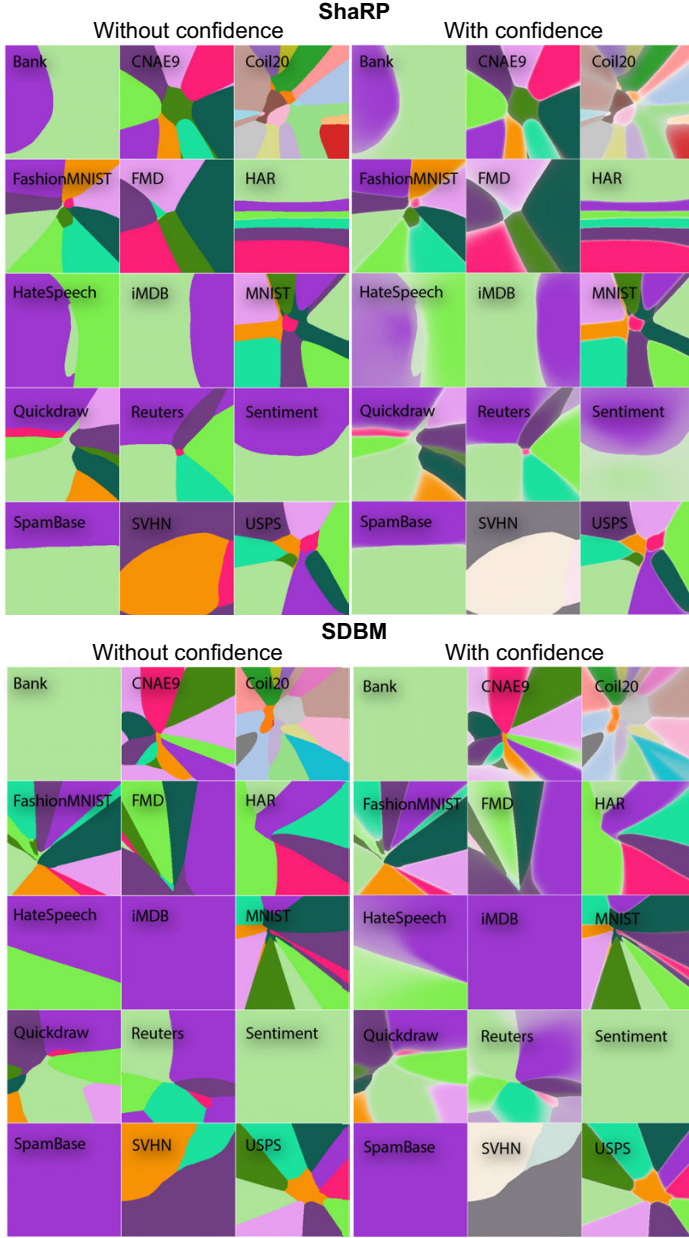


Figure 4.12: DBMs created by ShaRP and SDBM for all studied datasets, visualized without and with saturation-encoded confidence. The classifier is frozen per dataset. Immediately noticeable are the 2-class cases where SDBM produces degenerate DBMs (Sentiment, SpamBase datasets). We also see SSNP’s starburst pattern leaking into SDBM (Fashion-MNIST, MNIST datasets), which ShaRP avoids.

Bank and SpamBase). We believe this is due to a common practice in Deep Learning models: using a single neuron in the output layer of the classification network instead of naively using two, one per class. This absent neuron would indeed be redundant for classification since the output layer’s values must sum to one (as they form a probability distribution). We argue that this unnecessarily and strongly limits the power of SDBM to learn projection and inverse projection functions that place elements of different classes in significantly different regions of space. ShaRP does not possess that limitation, since it uses explicitly *two* neurons in its output layer for projections where only two classes of points exist.

Additionally, the starburst visual signature of SSNP leaks through SDBM, generating DBMs that have long, line-like regions (see datasets FashionMNIST, FMD, MNIST). ShaRP *might* generate line-like regions in DBMs as well, usually when the training process makes the learned clusters stack vertically or horizontally (see dataset HAR), but this effect is less frequent and the regions are not as thin as those in SDBM, which poses a smaller hindrance to understanding the visualization. The above show that ShaRP’s regularizing clusters in projections also regularizes the generated DBMs, while preserving the between-class structures as demonstrated in Section 4.3.4.

It is worthy of note that ShaRP’s classifier head (see Fig. 4.1) is able to predict classes directly for 2D points, thereby “painting” a DBM, *i.e.*, computing the labels that lead to colors for each DBM pixel. However, this classifier runs *directly on projected space coordinates*. In order to classify a new, unseen data point, this point must be projected by ShaRP and then classified, a process through which information may be lost.

In Sec. 4.2.3, we showed that ShaRP can create projections on the S^2 sphere, which can better show certain neighboring relations between sample clusters. The same idea applies to decision maps. Figure 4.13 shows decision maps created with ShaRP for a Multilayer Perceptron classifier trained on the Quickdraw and MNIST datasets. We see, just as we do for the 2D counterparts (Fig. 4.12), different decision zones of the classifier, with an additional degree of freedom given by the wrap-around nature of the S^2 space. The decision zones naturally enclose the training samples (small white dots) and have decreasing confidence as one approaches their boundaries. Note that our goal here to create DBMs which better capture the underlying neighborhoods of high-dimensional data points (which we achieve by using the S^2 sphere) is related conceptually to goals of preserving the data topology via projections (Doraiswamy et al., 2021). The key difference is that we aim to generate DBMs, which involve *extrapolating* a projection from a given dataset, whereas Doraiswamy et al. (2021) focuses on preserving topology as defined by a given dataset.

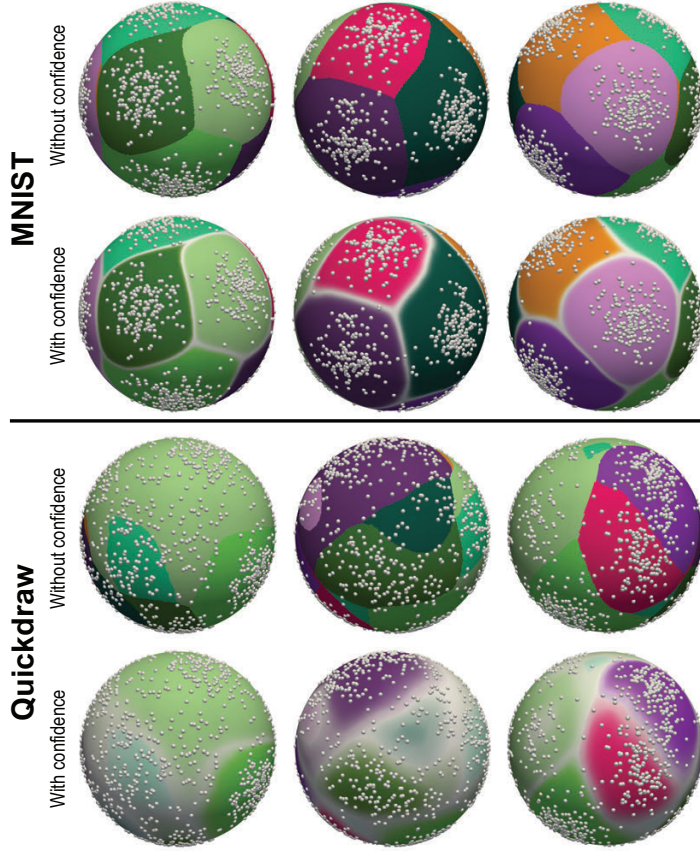


Figure 4.13: Decision maps for the Quickdraw and MNIST datasets using the S^2 space. Bottom row maps classifier confidence to color saturation. White dots show training samples. Compare with Fig. 4.7.

4.3.6 Computational performance

Figure 4.14 shows how ShaRP fares compared to other projection techniques *vs.* computational time. Tests were run on a PC with an AMD Ryzen 9 5900HX 3.3GHz 8-core processor and an NVIDIA RTX 3080 GPU. ShaRP is much faster than t-SNE (5 to 10 times speedup) and Isomap (5 to 6 times speedup). It is also faster than AE, UMAP, and only slightly slower than SSNP, its predecessor. The used batch size (256 data points) is largely responsible for ShaRP’s speed and, according to our experiments, does not negatively impact the quality metrics of the obtained projections, meaning it could be increased by a user whose priority is fast iteration. For ease of comparison, we use evaluation settings similar to those in other works advocating analogous Deep Learning architectures (Espadoto et al., 2019a,

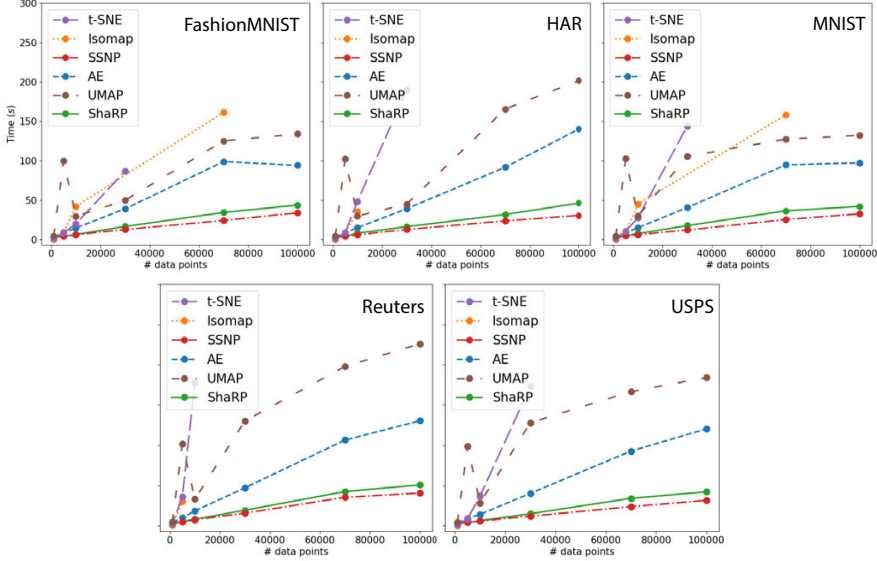


Figure 4.14: Times for a call to `fit_transform` (*i.e.*, both training and inference) for all studied projection techniques, five datasets. Runs were stopped after 5 minutes. We see that ShaRP is faster than all other tested DR techniques, being only marginally slower than SSNP, which does not offer visual pattern control. See Section 4.3.6.

2021a, 2020; Oliveira et al., 2023a). We refer the reader to Espadoto et al. (2020); Oliveira et al. (2023a) for experiments on how the batch size affects performance for a wide variety of datasets.

4.4 DISCUSSION

We next discuss several key aspects of our proposed method.

Genericity: ShaRP can handle high-dimensional datasets of any nature (*e.g.*, text, images, or tabular) as long as the data values are quantitative, as such data directly works with autoencoders. One-hot encoding or Categorical Variational Auto-Encoders (Morton et al., 2021) can overcome this limitation with only slight adaptations to our network architecture.

Inverse projections: Since using an autoencoder architecture, ShaRP automatically provides the inverse projection P^{-1} from the embedding to the data space at no additional cost or complexity.

Scalability: Since based on a deep learning architecture, ShaRP’s running time is linear in the number of samples and dimensions and considerably faster than classical methods such as t-SNE, Isomap, or UMAP (see

Sec. 4.3.6). After training, ShaRP’s operation is deterministic and also has the out-of-sample property, *i.e.*, it can project unseen samples along the training ones. This allows its simple usage for constructing decision maps, for which the inverse projection P^{-1} must be applied to unseen points (pixels).

Shape and space control: ShaRP allows controlling the shapes of similar-sample clusters (provided by (pseudo-)labels) to match ellipses, triangles, rectangles, or more generally any convex polygon. Doing this can decrease certain projection quality metrics as compared to using ‘free’ projections which do not constrain cluster shapes. We see this as an application-dependent trade-off that users can settle based on tuning a single, simple, parameter (see Sec. 4.2.2). As explained at the opening of this chapter, this trade-off means that, if users select, say, ShaRP with rectangles, they will (a) get a projection containing mainly rectangles and no other patterns; and (b) know that these patterns are due to their choice. In contrast, when using other DR techniques, one can expect a wide set of patterns in the projection (also depending on the technique’s parameter values). We argue that this can be confusing since (a) one has no clear control of the appearing patterns; and, more importantly, (b) the spontaneous appearance of such patterns can wrongly suggest they actually depict data patterns (Wattenberg et al., 2016). We remove such uncertainties by allowing the user to control the visual patterns in the projection. On the flip side, this means that if cluster shape patterns do exist in the data space, ShaRP will explicitly undo them and favor the user-chosen shape target instead. This is a relevant decision point for whether ShaRP is an adequate projection technique for a given task.

Shape-regularized projections can help visualization tasks. For example, rectangular shaped clusters ease the task of adding (rectangular) thumbnails or similar annotations atop of the respective clusters (see Fig. 4.3).

Independently on the selected cluster shape, ShaRP can project data to Euclidean spaces (like the 2D plane) or spherical ones (like $\mathcal{S}^2$). This is an additional global degree of control of the projection shape.

Decision maps: ShaRP can trivially create decision maps of any classification model trained on its input data samples. Compared to other decision map visualizations (*e.g.* SDBM or DeepView), ShaRP offers the additional aforementioned control of shape and space. Simply put, if this control enables one to generate a good *projection* for understanding a dataset, the same control (and projection) can be used to create a decision map for a classifier trained on that dataset.

Limitations: While flexible, ShaRP also has several limitations.

The sampling schemes we presented in Sec. 4.2 support shaping clusters into ellipses and convex polygons. A wider variety of shapes can

be obtained by devising new sampling schemes. However, obtaining log-probabilities for samples of complex sampling schemes can be computationally intensive.

As many other DR methods relying on clustering, *e.g.*, SSNP (Espadoto et al., 2021a; Oliveira et al., 2023a), ShaRP relies on a reasonable estimate of the number of clusters to use if no ground-truth labels exist. If ground-truth labels exist, the problem is solved — one either uses them directly or sets the cluster count to the number of different classes. Evidently, the quality of the clustering used in this case is important as well. This is a standard limitation of similar DR methods that use clustering.

More importantly, the issue of shape control raises some hard questions. On the one hand, no control allows a projection technique to *freely* place samples to optimally follow the data values. Yet, as we argued, this also allows the technique to enforce its own ‘visual signature’ which typically does not depend on the data but on the technique’s design, so it can be visually misleading. On the other hand, controlling shapes largely removes this algorithmic bias. Yet, this puts the responsibility of selecting ‘good’ shapes to further interpret the data on the user. All taken into account, we believe the latter option to be better. Yet, further studies *e.g.* involving user experiments are needed to see how shapes affect the perception of a ShaRP projection.

Another open question is how to control ShaRP in *intuitive* ways. So far, we have shown that the user can specify the shape of visual patterns to emerge in a projection, while taking some loss in projection quality in return. Yet, users may want to tune this, *e.g.*, specify how much they want to control shapes *vs* how much quality loss they agree with, beyond setting the ω parameter. A way forward would be to show local projection errors for ShaRP (following *e.g.*, Martins et al., 2014; Aupetit, 2007) and provide users with interactive tools to tune such errors *vs* the desired visual cluster shapes. A separate interesting open issue is how to quantify and display how well ShaRP enforces the desired visual shapes.

4.5 CONCLUSION

We have presented ShaRP, a new method for creating projections of high-dimensional data. ShaRP leverages variational autoencoders and (pseudo)labeled datasets to allow users to control the shapes of clusters created in Euclidean or spherical spaces. Alongside this, ShaRP keeps desirable features of projection techniques such as genericity, computational scalability, ease of use and implementation, and out-of-sample ability.

Several future work directions are possible. First, exploring new shape as well as space constraints would allow users an even greater control of the resulting projections. This can prove useful to *e.g.* create projections onto specific 2D or 3D manifolds that match known priors of the data

distribution. Secondly, we aim to study sampling schemes that naturally encourage further visual aspects of interest of projections, *e.g.*, cluster separability. Finally, we aim to explore applying ShaRP recursively to create an interactively navigable hierarchy of clustered data in projections.

We have so far presented two novel projection techniques: APPA, which approximates a reference projection, and ShaRP, which creates its own projection with controllable visual signature. To evaluate those, we resorted to Projection Quality Metrics, showing that the techniques produce sound representations of the data they project. We suspect, however, that these metrics have fundamental limitations. We ponder whether we can always trust that high metric values ensure that a good projection has been produced, since such metrics are essentially summary statistics of the DR process. In Chapter 5, we provide evidence of the fallibility of Projection Quality Metrics.

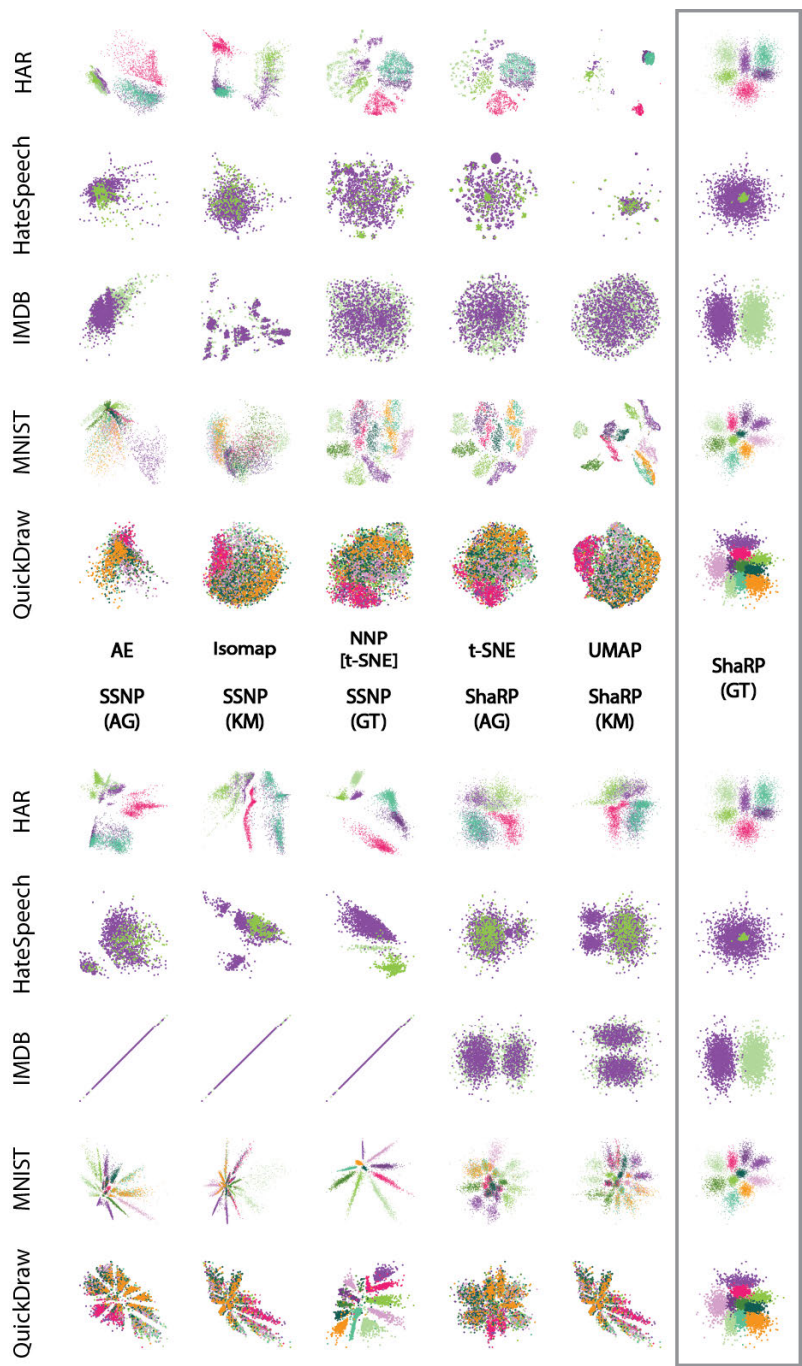


Figure 4.15: Continuation of Fig. 4.8.

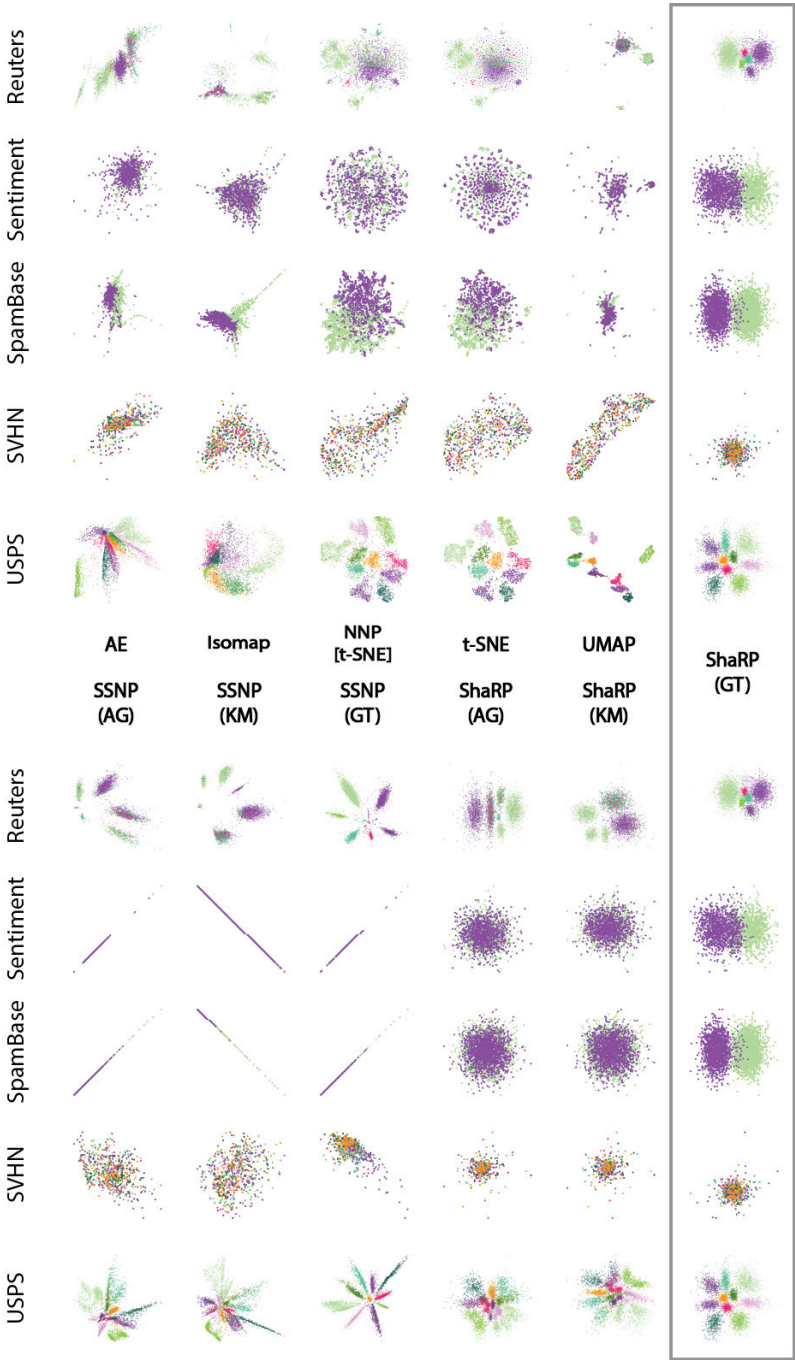


Figure 4.16: Continuation of Fig. 4.8.

LIMITATIONS OF PROJECTION QUALITY METRICS

A existência humana é, porque se
fez perguntando, a raiz da
transformação do mundo.[†]

Por uma pedagogia da pergunta

PAULO FREIRE E ANTONIO FAUNDEZ

5.1 INTRODUCTION

Dimensionality Reduction (DR) scatterplots encode data aiming to create *visual patterns* — e.g., grouping, spacing, proximity, outliers — that map related aspects for the data distribution. In general, no DR method can map data with a high intrinsic dimensionality to a 2D scatterplot while perfectly preserving all data structure. Projections can both *fail* to show true data patterns and also *create* visual patterns that do not represent data patterns — that is, at a high level, distort the data depiction (Nonato and Aupetit, 2018)¹.

Projection quality metrics (PQMs), introduced in Sec. 2.4, aim to provide a quantitative, objective, way to measure how well visual patterns in a projection map data patterns. Projections with high PQM values are deemed to faithfully reflect data patterns, so, usable for visual exploration; projections with low PQM values are discarded since they can be misleading.

In this work, we test the extent to which the above properties of PQMs are valid. Specifically, we show that several commonly-used quality metrics in DR fall short of gauging how well a projection captures data patterns. In detail, we show that projections having poor PQM values demonstrably capture poorly the data structure and thus should not be further used; however, we also show that it is possible to create projections with (very) high PQM values which show random, or even misleading, visual patterns which are not ground into data patterns. At a high level, our work shows that, much like summary statistics in exploratory data analysis, PQMs are *necessary* but *not sufficient* to characterize a faithful data depiction (Anscombe, 1973; Matejka and Fitzmaurice, 2017).

¹ This chapter is based on the paper “Necessary but not Sufficient: Limitations of Projection Quality Metrics” (Machado et al., 2025b).

[†] “Human existence, because it came into being through asking questions, is at the root of change in the world.”

To the above ends, we develop an algorithm that generates ‘fake’ projections containing arbitrary noise-like visual patterns scoring close to, or even higher than, projections created using standard DR techniques. We post-process our fake projections to introduce even more controlled visual patterns and still score high in quality metrics. While users can dismiss noise-like fake projections as unnatural, our post-processed fake projections have higher potential to mislead users to believe that the shown visual patterns actually represent data ones. We next show that it is easier to ‘fool’ certain metrics than others, and thereby propose a ranking of PQMs in terms of their ability to gauge a good projection. Our ‘fooling’ pipeline provides evidence to answer our RQ3.

5.2 BACKGROUND

We repeat here, for the reader’s convenience, the notations used throughout this chapter; for a full introduction, see Sec. 2.2.

A dataset $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_m\}$ is a collection of m data points in a n -dimensional space ($\mathbf{x}_i \in \mathbb{R}^n$). Points can be optionally paired with labels $c_i \in \{1, \dots, K\}$. We denote the label set of $\mathbf{X}$ by $C = \{c_1, \dots, c_m\}$ and the average of a set of N scalars by $\bar{\mu} = \frac{1}{N} \sum_i \mu_i$. Let $\mathbf{Y} = P(\mathbf{X}) = \{\mathbf{y}_1, \dots, \mathbf{y}_m\}$ be a *projection* of $\mathbf{X}$, where $\mathbf{y}_i \in \mathbb{R}^q$ with $q \ll n$. We next restrict ourselves to $q = 2$ since we consider projections visualized as 2D scatterplots. Except where stated otherwise, our scatterplots show class labels by point colors. Finally, we denote by $\mathcal{M}(\mathbf{X}, P(\mathbf{X})) \in \mathbb{R}_+$ the value of a quality metric which aims to gauge how well the scatterplot $P(\mathbf{X})$ captures the data structure of $\mathbf{X}$.

5.2.1 What is a good projection?

As discussed in Sec. 2.3, many projection techniques exist with different optimization aims and parameters, so one can easily create many different scatterplots $P(\mathbf{X})$ from the same data $\mathbf{X}$.

The currently accepted way to characterize the quality of such a projection scatterplot is to compute one or several so-called projection quality metrics (PQMs) which we introduced in Sec. 2.4. As explained there, PQMs have several advantages for assessing projections: they are quantitative (and typically deterministic); they can be computed easily, rapidly, and automatically. They allow a simple ranking of projections with respect to quality (Espadoto et al., 2021b): If P_1 scores higher than P_2 on several such metrics, then P_1 is found better than P_2 (relative assessment); the closer metric values are to the maximum, the better a projection is (absolute assessment). Such criteria help filter out or rank projections before users examine them (Johansson and Johansson, 2009; Huber, 1985; Wang and Mueller,

2017) and to holistically evaluate projection techniques (Morariu et al., 2023; Espadoto et al., 2021b).

Yet, we also made the case in Sec. 2.5 that PQMs, by themselves, are not sufficient to characterize what a good projection is. Specifically, we mentioned several limitations in how current PQMs gauge the ability of a projection to consistently map given *data patterns* to given *visual patterns*. We refine our analysis from Sec. 2.5 below to elicit several more specific factors that contribute to the observed limitations of current PQMs:

Level of detail: Current PQMs measure how P works *locally* or *globally* (see Sec. 2.4.1). Using only these two levels is not enough since humans examine scatterplots (or actually any image) over *many* scales (Albuquerque et al., 2011; Sedlmair and Aupetit, 2015; Espadoto et al., 2021b). Most PQMs model pattern scale by the size of k -nearest neighborhoods in data or projection spaces; or by clustering the data or projection, *e.g.*, by k -means and analyzing the resulting so-called motifs (Shao et al., 2016). Such relatively simple approaches cannot capture the full richness of pattern *shapes* that humans see, and reason about, when viewing a projection scatterplot — even if we were willing to spend the high price of computing PQMs for a wide set of k values (Martins et al., 2015). In fact, finding *all* such patterns in a plot Y is an open problem. Separately, we lack ways to find which data patterns are *missing* from a projection Y . Previous work has provided evidence that some quality metrics — most likely due to their above-mentioned dependence on the scale on which they are computed — can produce *false positives* when evaluating projections (Sedlmair et al., 2012).

In our work in this chapter, we will show that we can add to a projection salient, but arbitrary, visual patterns, on different scales, without changing its computed PQMs. This proves our point that current PQMs are not enough to fully capture projection quality.

Perception studies: Several studies have been conducted to find which quality metrics better align with how humans perceive projections (Sips et al., 2009; Sedlmair and Aupetit, 2015). This approach typically uses a correlation proxy: If users are more likely to pick a projection when a given metric is high, one may infer that such a metric aligns well with the human processing of projections. Such work is often task-based, *i.e.*, uses a scatterplot to solve a given problem or answer a given question. Such tasks include the identification of class clusters and their visual separation (Etemadpour et al., 2015; Sedlmair and Aupetit, 2015). Tatu et al. (2010) studied the perceived quality of 2D projections based on cluster separability and cluster density. Rensink and Baldrige (2010) studied how correlation is perceived in similar scatterplots. Other authors studied metrics that capture scatterplot perceptual similarity (Pandey et al., 2016; Dang and Wilkinson, 2014; Albuquerque et al., 2011). While useful to understand

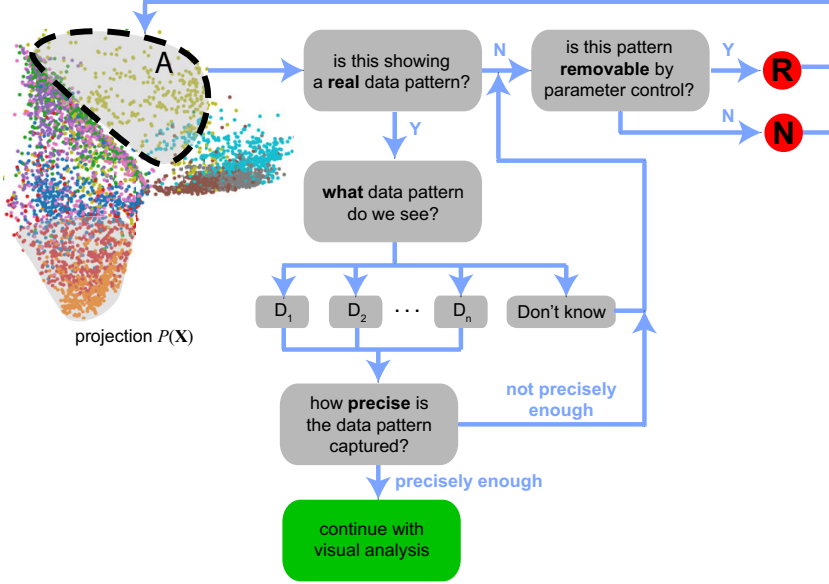


Figure 5.1: Visual analytics pipeline for the construction, assessment, and interpretation of projection scatterplots.

how humans interpret visual patterns in scatterplots, such studies do not answer how well these visual patterns actually capture *data patterns*.

Continuity: A metric that aims to rank some phenomenon on a *continuous* scale should itself be continuous (Allen and Yen, 1979). This property has been studied earlier under the name ‘stability’ for treemaps (Vernier et al., 2020b), decision maps (Oliveira et al., 2022), projections (Vernier et al., 2020a; Bredius et al., 2022), and inverse projections (Espadoto et al., 2023a), to mention only a few applications related to our context of high-dimensional data visualization. In our context, continuity means that small changes Δ in a projection should imply small changes in the PQM of that projection:

$$\mathcal{M}(\mathbf{X}, P(\mathbf{X}) + \Delta) \approx \mathcal{M}(\mathbf{X}, P(\mathbf{X})) + \mathcal{O}(\|\Delta\|).$$

The further we are from this goal, the less can $\mathcal{M}$ tell something *useful* about $P(\mathbf{X})$. Indeed, if $\mathcal{M}$ strongly changes while $P(\mathbf{X})$ stays about the same, then its values cannot characterize $P(\mathbf{X})$ robustly — see the work of Wang et al. (2020). Conversely, if $P(\mathbf{X})$ changes a lot (large Δ), then users will see different visual patterns in it, and in turn infer different data patterns. However, if $\mathcal{M}$ gives about the same value for the changed projection, then it will be wrongly labeled as ‘showing the same’. In our work, we will show that we can achieve the latter for many existing PQMs, which questions their ability to capture projection quality.

5.2.2 How to construct a good projection?

Given the aforementioned challenges regarding the assessment whether a projection scatterplot does capture well certain data patterns, how can we make sure this is the case — and what to do when this is not the case? This is, by no means, an easy question to answer in general.

Figure 5.1 sketches a — necessarily simplified — outline of how the above question is answered in typical visual analytics practice. Given some dataset X , one first constructs a projection scatterplot $P(X)$, given some suitably chosen technique P and hyperparameter settings. As starting points for this, one can use *e.g.* techniques and parameter settings which earlier evaluations have outlined as performing well for general data-analysis tasks, *e.g.*, t-SNE, UMAP, or MDS (Espadoto et al., 2021b).

Next, the analyst locates some salient visual patterns present in $P(X)$, such as the pattern marked ‘A’ in Fig. 5.1. One key question that comes next is whether this visual pattern shows an actual, real, data pattern or is an artifact of the working of P . One way to reduce the probability of the latter is to control the visual signature of projections such as explained in Chapter 4. If the conclusion is that the visual pattern A is not representing any actual data pattern, it is best if this pattern is removed from the visualization. This can be done by rerunning P with different parameter settings (option ‘R’ in Fig. 5.1); if this is not possible, one can try using other projection techniques instead of P (option ‘N’ in Fig. 5.1).

If the visual pattern under study is deemed to map actual data, a subsequent question is which actual data pattern this encodes. Many answers are possible here, denoted by the options $D_1, \dots, D_n$ in the figure. Examples hereof are the earlier-mentioned clusters of densely packed points with similar values for a dimension, clusters with different point densities, overlapping point clusters having different ground-truth labels, or outlier points far from the scatterplot bulk (see Sec. 2.5). Alternatively, the analyst can decide that the visual pattern under study maps some data pattern that they cannot understand. In the former case, the analysis proceeds as described below; in the latter case, it is advisable to recompute the projection to lead to more interpretable patterns.

If the visual pattern is traceable to a known data pattern, the next question one wants to answer is how precise is the said data pattern captured. Projection quality metrics can be used to this end — assuming, of course, that suitable ones exist to this end. If the result of this analysis indicates that the data pattern under study is too poorly captured for the goals of the current analysis, recomputation of the projection is the best option. In the opposite case, the projection scatterplot can be analyzed effectively (see green box in Fig. 5.1).

While, as already mentioned, Fig. 5.1 does not aim to provide a fully detailed rendering of how all visual analytics pipelines involving projec-

tion scatterplots operate, it makes a few key points for our work. First and foremost, it shows that interpreting projection scatterplots fundamentally relies on the *identification*, *quantification*, and *characterization* of visual patterns present in such scatterplots. Secondly, it shows that projection quality metrics (PQMs) are, to date, the only objective instrument we have in telling whether such visual patterns map (accurately) some data pattern. However, as discussed earlier in Sec. 5.2.1, current PQMs cannot, in general, fully characterize the wealth of visual patterns present in a projection. It follows that PQMs can *mislead* the analyst in a safe sense of security or a wrong interpretation of visual patterns. Our work next precisely confirms this hypothesis.

5.3 FOOLING PROJECTION QUALITY METRICS

While PQMs have several limitations (Sec. 5.2.1), projections are typically assessed by using precisely such metrics — not surprisingly given that one does not avail, to date, of better instruments for this task. Extensive evaluations have shown that projections scoring *low values* of PQMs are practically useless for data exploration (Espadoto et al., 2021b). This is not surprising: If a projection cannot preserve even the *basic* patterns current PQMs capture, then it is likely a poor projection.

However, the converse of the above point is unclear: *Do high values for PQMs mean that the visual patterns in a projection map data patterns clearly and accurately?* We will show that this is not the case. For this, we propose a systematic procedure to create projections that contain a scale of diverse visual patterns, some random and some showing misleading structures, *all of them uncorrelated to the underlying projected data*, and still obtain high PQM values. Just like summary statistics for data, we make the point that PQMs do not tell the whole story about a projection’s ability to map data patterns (Matejka and Fitzmaurice, 2017; Anscombe, 1973). While PQM’s use as exclusion criteria is justified — low PQM values mean poor projections — the converse does not generally hold.

Our approach works as follows (see also Fig. 5.3): Given a dataset $\mathbf{X}$ and its projection $\mathbf{Y} = P(\mathbf{X})$ computed by any user-chosen technique P , we aim to modify, or ‘fool’, $P(\mathbf{X})$ in random or determined ways while keeping $\mathcal{M}(\mathbf{X}, P(\mathbf{X}))$ constant or even increasing it. This yields ‘fake’ projections of $\mathbf{X}$ that show arbitrary visual patterns in $\mathbf{Y}$ with no PQM cost decrease; hence, the said PQMs are not good to gauge how well visual patterns in $\mathbf{Y}$ capture data patterns in $\mathbf{X}$. One way to distort $\mathbf{Y}$ while increasing PQM $\mathcal{M}$ would be to use some sort of gradient ascent: compute $\nabla_{\mathbf{Y}} \mathcal{M}(\mathbf{X}, \mathbf{Y})$ and move points in $\mathbf{Y}$ according to it. Yet, many PQMs are not differentiable, so we cannot directly compute $\nabla_{\mathbf{Y}} \mathcal{M}$. To address this, we:

1. approximate $\mathcal{M}$ with a continuous model differentiable with respect to $\mathbf{Y}$: a feed-forward neural network Q_θ with parameters θ minimizing a regression loss;
2. use (as approximation target μ) the local version of $\mathcal{M}$ (Eqn. 2.1) as a proxy, since we need one target value per $(\mathbf{x}_i, \mathbf{y}_i)$ pair during training, whereas a global metric such as Shepard goodness yields one target value for the entire $P(\mathbf{X})$. This limits our method to target metrics of the type given by Eqn. 2.1. The loss function for Q_θ then reads

$$\mathcal{L}_Q(\theta) = \frac{1}{n} \sum_{i=1}^n (Q_\theta(\mathbf{x}_i, \mathbf{y}_i) - \mu_i(\mathbf{X}, \mathbf{Y}, C))^2. \quad (5.1)$$

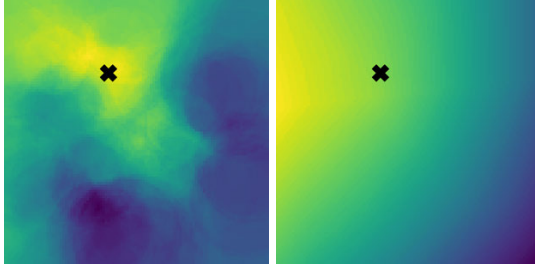


Figure 5.2: Left: Each pixel $\mathbf{p}$ shows the $\mathcal{M} = \text{Trustworthiness}$ value of a projection obtained by moving a single point (marked X) of a reference $P(\mathbf{X})$ to $\mathbf{p}$. Here, $P = \text{t-SNE}$ and $\mathbf{X} = \text{MNIST}$. Right: predicted Q_θ after training to approximate Trustworthiness for this projection. While most of the true behavior of $\mathcal{M}$ is not captured by Q_θ , the approximation captures correctly the high/low value zones.

This learning setup does not capture a PQM in a way that generalizes to new projections of the same data. Since PQMs are typically discontinuous or at least non-differentiable, changes to one $\mathbf{y}_i$ do not smoothly change the value of $\mathcal{M}$. We do not propose, and do not need, a continuous approximation of PQMs and refrain from using Q_θ as a *true* approximation to $\mathcal{M}$. Figure 5.2 shows this for the $\mu = \text{Trustworthiness}$ metric. We see that μ and Q_θ have different variations when we change $P(\mathbf{X})$. Despite this, the approximation Q_θ is still useful for our purposes (see Sec. 5.3.1).

5.3.1 Pattern-destructive phase

We set out to create projections $\hat{\mathbf{Y}}$ which are evidently poor but still score well according to a target metric μ by using a modified version of NNP (Espadoto et al., 2019a). NNP outputs a projection $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$ by

training a neural network P_ϕ to mimic a reference projection P of a subset $\mathbf{X}' \subset \mathbf{X}$. NNP's loss function is a simple Mean Squared Error

$$\mathcal{L}_{\text{NNP}}(\phi) = \frac{1}{|\mathbf{X}'|} \sum_{i=1}^{|\mathbf{X}'|} \|P_\phi(\mathbf{x}'_i) - P(\mathbf{x}'_i)\|^2. \quad (5.2)$$

Beyond what NNP does, we wish to create a projection that has high values of one or more PQMs *and* remove visual patterns that exist in $\mathbf{Y}$ (thereby creating type-I errors). For this, we pick a metric μ to fool and use Q_θ , trained to approximate μ , as an extra loss to *maximize*. This extra loss can destabilize training, likely due to our approximator inability to fully capture μ . To fix this, we add an auxiliary learning goal that reconstructs $\mathbf{X}$ from the learned projection $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$, following (Machado et al., 2024e; Makhzani et al., 2015). For this, we augment NNP with a network $\mathcal{I}_\psi : \mathbb{R}^2 \rightarrow \mathbb{R}^D$ with parameters ψ outputting $\hat{\mathbf{X}} = \mathcal{I}_\psi(P_\phi(\mathbf{X}))$ and minimizing a *reconstruction* loss. The complete loss to be minimized reads

$$\begin{aligned} \mathcal{L}_{\text{Fooler}}(\phi, \psi) = & -\frac{1}{n} \sum_{i=1}^n \left(Q_\theta(\mathbf{x}_i, P_\phi(\mathbf{x}_i)) - \kappa \|P_\phi(\mathbf{x}_i) - \mathbf{y}_i\|^2 \right. \\ & \left. - \ln(1 + \|\mathcal{I}_\psi(P_\phi(\mathbf{x}_i)) - \mathbf{x}_i\|^2) \right), \end{aligned} \quad (5.3)$$

where $x \mapsto \ln(1 + x)$ squashes down high values of the reconstruction loss. We set $\kappa = 10$ for the first 10 training epochs and next set it to zero. This makes the model first learn the reference projection, much like NNP; next, the model can freely modify the learned projection to get higher μ values as predicted by Q_θ . This process naturally generates projections with poor visual patterns, since point positions are optimized independently, and according to Q_θ , an approximation of limited accuracy. Figure 5.3 shows our main fooling pipeline.

The differentiability of Q_θ with respect to the projection $\mathbf{Y}$ enables guiding P_ϕ towards regions with higher Q_θ values. This is because gradients can flow *through* Q_θ into P_ϕ ; this would not happen if we used μ (or $\mathcal{M} = \bar{\mu}$) instead of Q_θ since μ is not differentiable. Indeed, the first term on the right hand side of

$$\frac{\partial \mu}{\partial \phi} = \frac{\partial \mu}{\partial \mathbf{y}_i} \Big|_{\mathbf{y}_i = P_\phi(\mathbf{x}_i)} \frac{\partial \mathbf{y}_i}{\partial \phi} \quad (5.4)$$

does not exist, while $\partial Q_\theta / \partial \mathbf{y}_i$ (using $Q_\theta \approx \mu$) is always defined.

After training, we run P_ϕ in inference mode with $\mathbf{X}$ as input to get our object of interest $\hat{\mathbf{Y}}$ and *discard* all neural networks. Figure 5.4 shows examples of our fooled projections $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$ for the HAR dataset and four quality metrics. The learned projections for Trustworthiness and Continuity are visually poor (Fig. 5.4a,c): Points collapse towards a scatterplot corner; class separation is low; yet, both target metrics suffer only minor penalties of

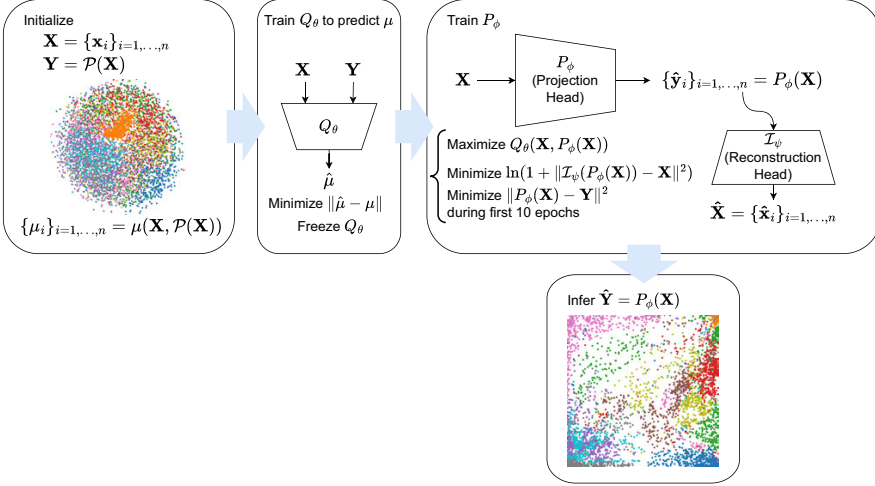


Figure 5.3: Architecture of our projection fooling method. From a given dataset $\mathbf{X}$ and a reference projection $\mathbf{Y} = P(\mathbf{X})$, we compute a local quality metric μ and train a neural network Q_θ to infer μ at all points $P(\mathbf{X})$. We then freeze Q_θ and use it, with $\mathbf{X}$ and $\mathbf{Y}$, to train the neural network P_ϕ to project $\mathbf{X}$ as to maximize Q_θ while minimizing a reconstruction error. We also force P_ϕ to learn the reference projection $\mathbf{Y}$ for the first 10 training epochs. After training, we generate the fooling projection $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$ and discard P_ϕ , $\mathcal{I}_\psi$, and Q_θ .

0.02 and 0.01, telling that they are *not* sensitive to the artificial patterns our fooling introduces. The learned projection for Neighborhood Hit is also degenerate (Fig. 5.4d) but less than (a) or (c); here, the target metric suffers a heavier penalty. Finally, when aiming to fool the Jaccard metric, we see the least degenerate projection of all (Fig. 5.4b); quality has now a heavy drop of almost 0.2.

5.3.2 Pattern-constructive phase

Figure 5.4 shows that P_ϕ creates projections with random visual patterns but with PQM values close to those of the ground-truth projection. Yet, one could argue that users might quickly feel that such projections are fabricated and, even if they have high quality values, dismiss them from further exploration. We next present three postprocessing techniques that modify fooled projections to keep high PQM values *and* contain visual patterns typically seen in good-quality projections that do not reflect the underlying data (thereby creating type-II errors).

Postprocessing aims to counteract what a salient aspect of $\hat{\mathbf{Y}}$ — the collapse of the scatterplot onto the projection boundaries (Fig. 5.4). We cannot

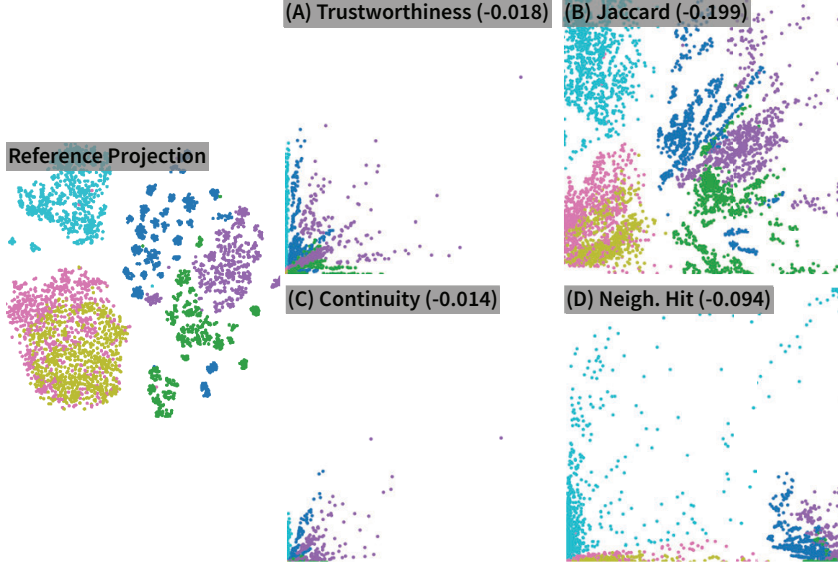


Figure 5.4: Fooled projections $\hat{Y} = P_\phi(X)$ for the HAR dataset starting from the same t-SNE projection P (left) for different target metrics μ . We annotate each projection with its target μ and the difference in $\mathcal{M} = \bar{\mu}$ using P as reference (in brackets). We see only slight losses in Trustworthiness and Continuity despite the projections looking degenerate. We see higher losses in Neighborhood Hit and Jaccard Similarity. For the latter case, the visual patterns we obtain are more in line with those usually expected in projections.

change $\hat{Y}$ arbitrarily since we want to affect PQMs as little as possible. This suggests some sort of ‘semi-rigid’ scatterplot transformation that preserves neighborhoods. Our postprocessing techniques all work by building a graph from the projection $\hat{Y}$ or from the dataset X (see Fig. 5.5) in different ways, as explained next.

Data Nearest Neighbors: We build a weighted k -nearest neighbors graph G_{DataNN} of X , with weights given by Euclidean distances between data points — we use $k = 7$ in our experiments.

Projection Nearest Neighbors: We build the unweighted k -nearest neighbors graph G_{ProjNN} of $\hat{Y}$ — we use $k = 7$ in our experiments. We use an unweighted graph as the distances in projection space are not necessarily meaningful after the fooling procedure.

Delaunay Triangulation: We build the Delaunay triangulation G_Δ of the projection $\hat{Y}$ forcing simplicial output.

On all these graphs, we apply 5 iterations of the standard force-directed layout (Fruchterman and Reingold, 1991) to tweak point positions in $\hat{Y} = P_\phi(X)$. For the first two graphs, we use the default ideal edge length

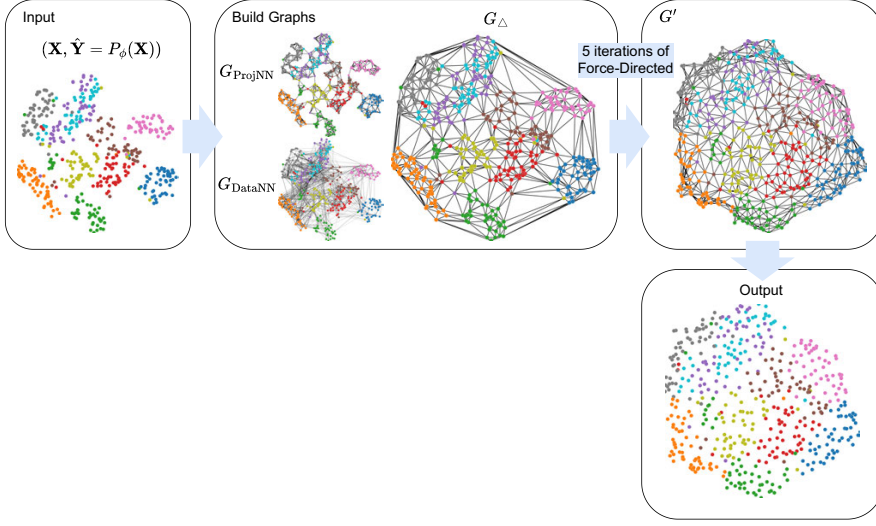


Figure 5.5: Architecture of our post-processing pipeline. Starting from a given dataset $\mathbf{X}$ and fooled projection $\hat{\mathbf{Y}}$, we build three different graphs — G_{ProjNN} , G_{DataNN} , and G_{Δ} . Each of those three graphs is then used to perform 5 iterations of a force-directed layout, producing three different outputs. To avoid clutter, we visualize here only the output using G_{Δ} .

$l = 1/\sqrt{n}$; for G_{Δ} , we set l to the 40th-percentile of pairwise projection distances.

Figure 5.6 shows the results of our three postprocessing methods for the HAR projection (Fig. 5.4c). Scatterplot points spread better as compared to the unprocessed version. The resulting scatterplots also appear more similar to typical projections in DR literature. Large-scale visual patterns appear, *e.g.*, circular structures which could fool users to believe that they are data-driven. Postprocessing only lightly affects the PQMs (see Fig. 5.7) and can even *increase* these. All in all, we see that introducing *significant* visual patterns that are unrelated to the data is not detected by the evaluated PQMs.

5.4 EVALUATION

We evaluate the ability of our fooling mechanism to create projections with high values of various PQMs. As many such PQMs exist, we focus next on metrics which are

- **easy to compute:** metrics should have few hyperparameters;

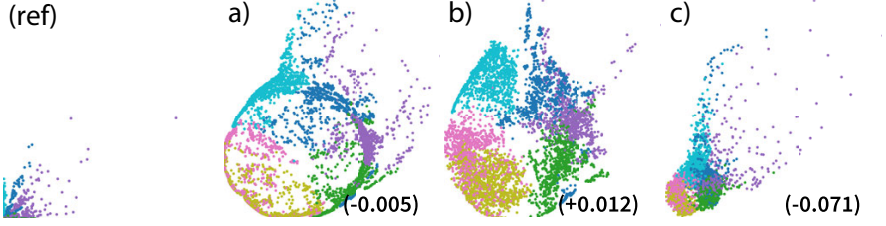


Figure 5.6: Postprocessing with (a) G_{ProjNN} , (b) G_{DataNN} , (c) G_{Δ} of the Continuity-fooling projection of the HAR dataset (ref). Annotations show the difference in Continuity *vs* the unprocessed output. While (c) shows a more important loss, (a) shows barely any change in Continuity. (b) shows an *increase* in Continuity and visual patterns similar to those typical in good projections.

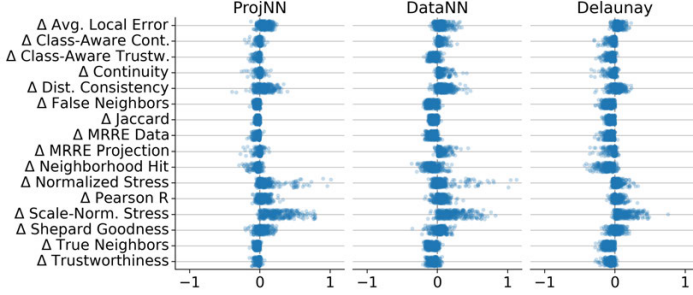


Figure 5.7: Effect of post-processing on studied quality metrics across our experiments. We measure the difference $\Delta M = M_{\text{Post}} - M^{\dagger}$ between metrics learned by our fooling pipeline $M^{\dagger}$ and their values after post-processing M_{Post} . Points are mostly concentrated around 0, meaning there is no strong (negative or positive) influence in terms of quality in the constructive phase of the pipeline.

- **deterministic:** for a given $\mathbf{X}, \mathbf{Y}$, and C , the metric value should be the same; non-deterministic metrics are less useful in practice for assessing the quality of a given study object;
- **common:** fooling a metric well-established in DR literature has arguably stronger impact than fooling an infrequently used one.

With these criteria, we selected 17 quality metrics (Tab. 7). Their definitions, value ranges, and optimal values are detailed in Sec. 2.4.2.

5.4.1 Experiment setup

Projections: We tested four commonly-used projection techniques, namely t-SNE (van der Maaten and Hinton, 2008), UMAP (McInnes and Healy,

Table 7: Projection quality metrics used in the evaluation.

Metric	Parameters	Introduced by
Average Local Error	—	Martins et al. (2014)
Continuity	k	Venna and Kaski (2006)
Class-Aware Continuity	k	Colange et al. (2020)
Distance Consistency (DSC)	—	Sips et al. (2009)
False (resp. True) Neighbors Rate	k	Martins et al. (2014)
Jaccard Similarity of Neighbor Sets	k	Jaccard (1901)
Mean Relative Ranking Errors	k	Lee and Verleysen (2009)
Neighborhood Hit	k	Paulovich et al. (2008)
Normalized Stress	—	Kruskal (1964)
Pearson Correlation of Distances	—	Geng et al. (2005)
Procrustes Statistic	k	Goldberg and Ritov (2009)
Scale-Normalized Stress	—	Smelser et al. (2024)
Shepard Goodness	—	Siegel and Castellan (1988)
Trustworthiness	k	Venna and Kaski (2006)
Class-Aware Trustworthiness	k	Colange et al. (2020)

2018), MDS ([Kruskal, 1964](#)), and Isomap ([Tenenbaum et al., 2000](#)). These techniques are chosen due to their wide adoption, readily-available implementation ([Pedregosa et al., 2011](#)), and diversity in terms of mathematical grounding and algorithm design.

Datasets: We used six datasets (Tab. 8) with a stratified subsample of at most 5000 samples from each dataset in all experiments.

Table 8: Datasets used for experimentation.

Dataset	# Dims. (n)	# Classes (K)
FashionMNIST (Xiao et al., 2017)	784	10
Human Action Recognition (Anguita et al., 2012)	561	6
MNIST (LeCun et al., 1998)	784	10
Reuters (Thoma, 2017)	5000	6
Spambase (Hopkins et al., 1999)	57	2
USPS (Hull, 1994)	256	10

Target metrics: For the *local* metric to fool (Fig. 5.3, Sec. 5.3), we explore four cases: Continuity, Jaccard set similarity, Neighborhood hit, and Trustworthiness, and also fooling all four metrics jointly. All these metrics have one hyperparameter, the neighborhood size k , which we set to $k \in \{1, 7, 21, 51\}$. In total, we consider 6 datasets $\times$ 4 projections $\times$ (4 + 1) fooling targets $\times$ 4 hyperparameter settings = 480 total experiments. For each setting, we measure the 17 chosen metrics — identically setting k when applicable — leading to 8160 measured values in total. Given this amount, we next show different kinds of aggregation and focus on specific instances. Further details are given in Appendix A.3. All results can be explored interactively [online](#) (Machado et al., 2024b) (see also Fig. 5.8).

Implementation details: We have a large experimentation space, so computing metrics must be fast. We implement the 17 studied metrics using TensorFlow (Abadi et al., 2016) as a PyPI (Machado et al., 2024c) package (for details, we refer to Machado et al. (2025a)). Our code is openly available (Machado et al., 2024a).

Hyperparameter settings: All our studies use the same hyperparameters. We train all our neural networks with a learning rate of 10^{-3} , mini-batches of 256 data points, and the Adam (Kingma and Ba, 2015) optimizer, 100 epochs (Q_θ), and 1000 epochs (P_ϕ together with $\mathcal{I}_\psi$). Full architecture details are given in Appendix A.3.2.

5.4.2 Studying the created projections

We start exploring by looking at the outputs generated by our fooling pipeline (fooled projections and their quality scores) and compare these with the reference P . Figure 5.9 shows a selected subset of such results. All experiments are available in the online material.

Figure 5.9a shows the projections $\mathbf{Y}$ used as reference by our fooling pipeline, each annotated with the value of the metric targeted for fooling. Columns (b) show the learned $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$ before post-processing. These projections have undesirable visual properties (point overlap, cluster collapse, and points with extreme x and/or y coordinates), so they clearly cannot preserve any meaningful data patterns — they actually show no human-digestible visual patterns overall. Yet, the evaluated metrics score most of these fake projections *highly* — see columns (b) — often *higher* than the reference projection. Columns (c-e) show the outcomes of our post-processing G_Δ , G_{ProjNN} , and G_{DataNN} . The post-processed projections all look more plausible, *i.e.*, contain visual patterns one typically sees in DR scatterplots. Yet, these patterns are *misleading*: For example, the plot for Isomap-MNIST-NH created by $G_{\text{ProjNN}}(\clubsuit)$ looks like a symmetric dumbbell while the ground-truth $\mathbf{Y}$ shows there is no such structure in MNIST;

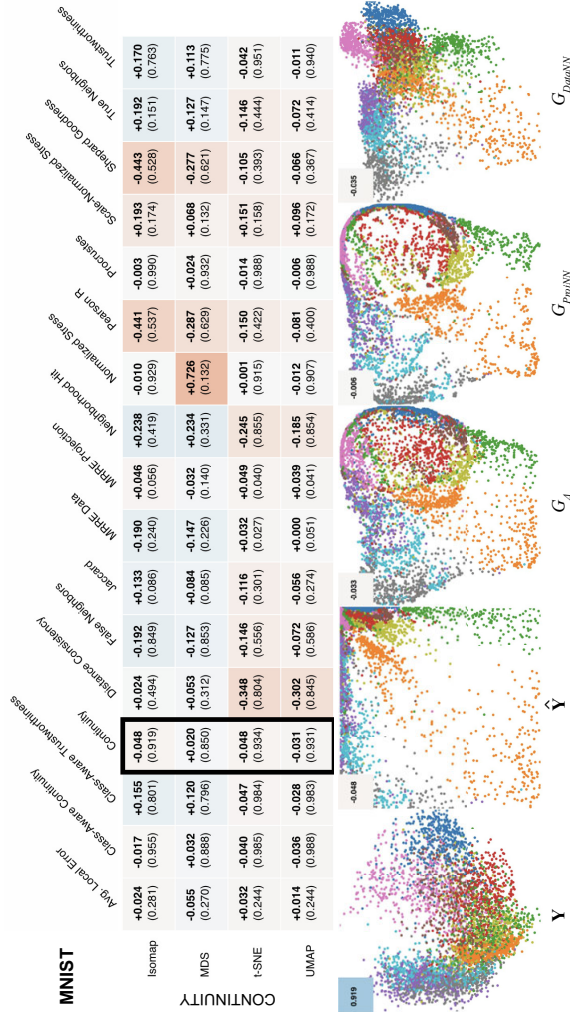


Figure 5.8: Exploratory online tool for studying projection fooling results. Top: Results of fooling Continuity (outlined in black) for studied DR techniques on MNIST. Table cells colored to show PQM gain resp. loss *vs* the reference projection Y (blue: gain; white: neutral; red: loss). Fooling yields practically identical Continuity, but also similar values for the other 17 non-targeted metrics. The main losses are for Stress (MDS), Distance Consistency (t-SNE, UMAP), and Pearson R and Shepard (Isomap). Bottom: reference Y , learned $\hat{Y}$, post-processed outputs G_{Δ} , G_{ProjNN} , G_{DataNN} , annotated with change in Continuity *vs* Y . Fooling produces random-like patterns ($\hat{Y}$) but also salient structures which have no meaning (G_{ProjNN} , G_{DataNN}).

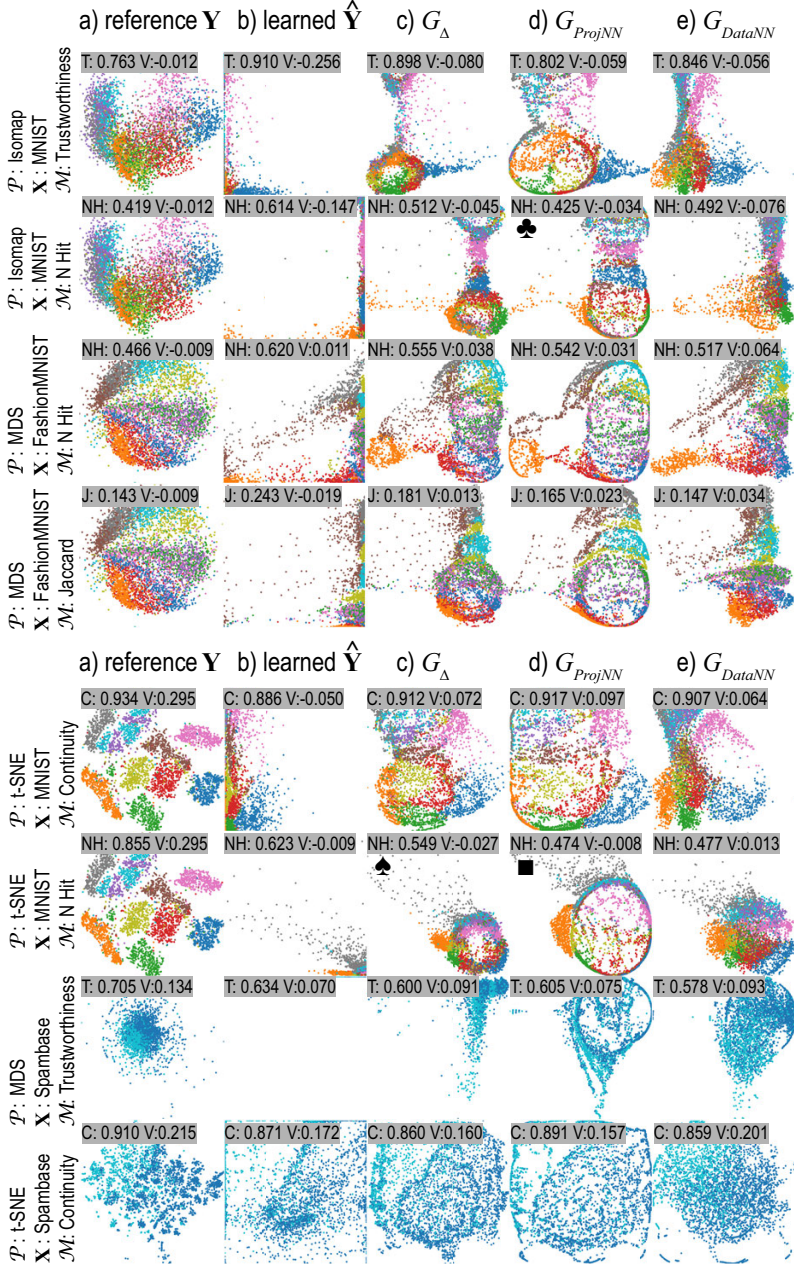


Figure 5.9: Samples of our fooling pipeline for different projection techniques P , datasets X , and target metrics M . Each projection is annotated with the value of the target metric M , as well as of CIVM (Liu et al., 2010), a VQM measuring cluster preservation (V in each plot). For further discussion on the elements marked with glyphs, see Section 5.4.2.

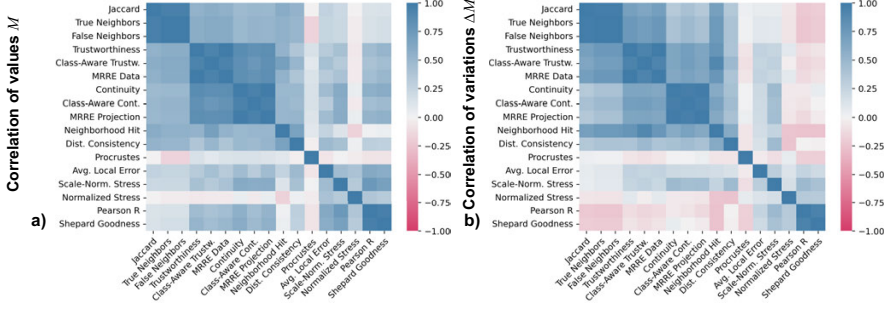


Figure 5.10: Correlations in metric values M (a) and value variations ΔM (b). Blue: metrics that vary in the same direction (a), resp. metrics showing the same fooling behavior (b). We use remapped values (see Tab. 9) such that positive metric values always mean high quality.

the plot for t-SNE-MNIST-NH created by G_{ProjNN} (■) shows a circular structure which, again, does not exist in MNIST; the plot for t-SNE-MNIST-NH created by G_{Δ} (♠) shows a cloud of outliers top-left which do not exist in MNIST. Post-processing weakly affects quality values, sometimes positively, sometimes negatively — another instance of insensitivity of the quality metrics to introducing arbitrary patterns into a projection. Indeed, of all post-processing approaches, only G_{DataNN} uses ground-truth data, and still does not always lead to a gain in quality metrics.

5.4.3 Correlations in fooling behavior

We next explore all metric values of the fooled projections $\hat{\mathbf{Y}}$ before post-processing (Sec. 5.3.2) to find which metrics tend to move the same way during fooling. For this, we measure all 17 metrics for a reference projection $\mathbf{Y} = P(\mathbf{X})$, obtaining $M = [\mathcal{M}_1(\mathbf{X}, \mathbf{Y}), \dots, \mathcal{M}_{17}(\mathbf{X}, \mathbf{Y})]$. We run our fooling pipeline with a fooling target $\mathcal{M}^\dagger$ and obtain $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$. We measure again all 17 metrics on $\hat{\mathbf{Y}}$, obtaining $M^\dagger = [\mathcal{M}_1(\mathbf{X}, \hat{\mathbf{Y}}), \dots, \mathcal{M}_{17}(\mathbf{X}, \hat{\mathbf{Y}})]$. The value $\Delta M = M^\dagger - M$ tells us how much fooling the *single* target $\mathcal{M}^\dagger$ affected *all* 17 metrics. Doing this for all experiment settings yields a set of 480 different $(M, M^\dagger, \Delta M)$. With these data, we set out to answer two questions:

1. Which metrics behave *similarly* across projections? Figure 5.10a answers this by showing the correlations in M .
2. Which metrics does our fooling affect *as a group*? That is, when fooling some $\mathcal{M}^\dagger$, which other metrics are consistently fooled too. Figure 5.10b answers this by showing the correlations in ΔM .

Table 9: Re-mapping metric values for comparison purposes. The best value in each scenario is shown in bold.

Range of $m = \mathcal{M}(\mathbf{X}, \mathbf{Y})$	Transformation	Range of $f(m)$
$[0, +\infty)$	$f(m) = -m$	$(-\infty, 0]$
$[a, b]$	$f(m) = (m - a) / (b - a)$	$[0, 1]$

Yet, our 17 metrics have values inconsistent in meaning: Stress and Procrustes range from 0 (best) to infinity (worst); Trustworthiness and Continuity range in 0 (worst) to 1 (best). This creates negative correlations where pairs of metrics would agree on improvements. To fix this, we re-map all metrics so higher values mean better quality (Tab. 9). Any increase $\Delta M_i > 0$ now means an increase in quality. In Figure 5.10a, blue (resp. red) show positively (resp. negatively) correlated metric values for the *reference* projections. That is, a blue cell for a metric pair tells that these metrics move in roughly the same way — a projection scoring, say, high for Trustworthiness also scores high on Continuity. We see that most of the metric pairs are positively correlated — meaning that high values (after re-mapping) consistently indicate high quality. This provides us the baseline for addressing question 2 as follows. Figure 5.10b shows how fooling a single metric affects the other metrics. Blue (resp. red) cells tell that our fooling produces similar (resp. opposite) variations in metric values. We see that, even though most experiments target fooling one metric, this affects other metrics as well. Metrics appear split into two types: The large blue block (top-left in Fig. 5.10b) shows metrics that can be easily fooled together — we can generate fake projections having large values of any metric in this block by fooling a single metric herein. More interestingly, the light red cells tell us that Normalized Stress, Pearson Correlation, and Shepard Goodness decrease when we try to fool any of the other 17 metrics. Conversely, fooling these three metrics will decrease some of the remaining ones. This means that using metrics from the two types *together* can provide a more robust evaluation of projection quality than using only metrics of a single type.

5.4.4 Ease of fooling

We next study how well our fooling can yield *higher metric values* while creating arbitrary patterns in projections. For all the metrics we can fool — Continuity, Jaccard, Neighborhood hit, Trustworthiness, and all jointly (Sec. 5.4.1) — we plot the distribution of the difference ΔM for all measured 17 metrics for all experiments (Fig. 5.11a). Distributions concentrated around zero tell that the respective metric (see names in the left labels) are

kept well irrespective which target metric we fool — *e.g.*, Class-Aware Continuity. Distributions skewed to the *right* tell that metric values increase after fooling — so, given our ‘higher is better’ remapping, fooling actually *increases* quality measured by that metric — *e.g.*, Trustworthiness and Jaccard. *Left*-skewed distributions tell that our fooling slightly loses quality as measured by that metric — *e.g.*, Distance Consistency and Normalized Stress. The same data is shown using box-plots in Figure A.11. Overall, these figures tell us that fooling any of the target metrics is quite successful, not only in the target values, but also in affecting all other metrics.

Figure 5.11b shows the same information grouped per *projection technique*. We do not see any strong bias of success or failure of fooling *vs* a technique. Yet, we see quality losses for stress metrics for MDS after fooling. This is expected: The MDS cost function aims to optimize *precisely* a formulation of stress. Finally, Fig. 5.11c shows the same information grouped per *dataset*. We do not see any strong bias of success or failure of fooling *vs* a specific dataset. The only exception is Spambase, where quality consistently decreases after fooling. We attribute this to the *low dimensionality* of the data ($D = 57$): the reconstruction head $\mathcal{I}_\psi$ uses a 3-layer neural network with 512 neurons in the last hidden layer, almost 10 times larger than D , leading to an overparameterized regime that causes data points to collapse to the same place in the projection.

Table 10: Quality Metric values (high value = high quality) for the subset of metrics found to be least correlated (see Sec. 5.4.5). Each row compares the value after fooling with the pre-fooling value (in brackets). We highlight the 2 strongest quality losses per row.

Dataset	Fooling target	P	Cont.	Trust.	True Neigh.	Pearson r	DSC	Neigh. Hit	Procrustes	Stress
MNIST	Trustw.	Isomap	0.870 (0.919)	0.910 (0.763)	0.292 (0.151)	0.588 (0.768)	0.272 (0.494)	0.609 (0.419)	-0.963 (-0.990)	-0.921 (-0.929)
			0.894 (0.931)	0.931 (0.940)	0.348 (0.414)	0.694 (0.700)	0.526 (0.845)	0.669 (0.854)	-0.977 (-0.988)	-0.880 (-0.907)
			0.903 (0.934)	0.935 (0.951)	0.338 (0.444)	0.725 (0.711)	0.515 (0.804)	0.675 (0.855)	-0.961 (-0.988)	-0.981 (-0.915)
		UMAP	0.729 (0.872)	0.632 (0.709)	0.130 (0.191)	0.653 (0.817)	0.483 (0.804)	0.619 (0.765)	-0.928 (-0.983)	-0.375 (-0.547)
			0.658 (0.878)	0.555 (0.851)	0.042 (0.402)	0.598 (0.648)	0.616 (0.782)	0.628 (0.780)	-0.979 (-0.992)	-1.193 (-0.554)
			0.640 (0.910)	0.574 (0.873)	0.046 (0.444)	0.591 (0.684)	0.790 (0.790)	0.641 (0.790)	-0.975 (-0.985)	-1.155 (-0.275)
Spambase	N. Hit	Isomap	0.921 (0.946)	0.943 (0.855)	0.334 (0.231)	0.631 (0.847)	0.669 (0.627)	0.736 (0.625)	-1.000 (-0.991)	-0.978 (-0.883)
			0.929 (0.960)	0.933 (0.962)	0.315 (0.461)	0.611 (0.738)	0.465 (0.949)	0.742 (0.928)	-0.963 (-0.995)	-0.863 (-0.837)
			0.916 (0.962)	0.926 (0.967)	0.343 (0.486)	0.748 (0.748)	0.604 (0.948)	0.737 (0.932)	-0.960 (-0.989)	-0.791 (-0.849)
		UMAP	0.921 (0.946)	0.943 (0.855)	0.334 (0.231)	0.631 (0.847)	0.669 (0.627)	0.736 (0.625)	-1.000 (-0.991)	-0.978 (-0.883)
			0.929 (0.960)	0.933 (0.962)	0.315 (0.461)	0.611 (0.738)	0.465 (0.949)	0.742 (0.928)	-0.963 (-0.995)	-0.863 (-0.837)
			0.916 (0.962)	0.926 (0.967)	0.343 (0.486)	0.748 (0.748)	0.604 (0.948)	0.737 (0.932)	-0.960 (-0.989)	-0.791 (-0.849)
USPS	Continuity	Isomap	0.921 (0.946)	0.943 (0.855)	0.334 (0.231)	0.631 (0.847)	0.669 (0.627)	0.736 (0.625)	-1.000 (-0.991)	-0.978 (-0.883)
			0.929 (0.960)	0.933 (0.962)	0.315 (0.461)	0.611 (0.738)	0.465 (0.949)	0.742 (0.928)	-0.963 (-0.995)	-0.863 (-0.837)
			0.916 (0.962)	0.926 (0.967)	0.343 (0.486)	0.748 (0.748)	0.604 (0.948)	0.737 (0.932)	-0.960 (-0.989)	-0.791 (-0.849)
		UMAP	0.921 (0.946)	0.943 (0.855)	0.334 (0.231)	0.631 (0.847)	0.669 (0.627)	0.736 (0.625)	-1.000 (-0.991)	-0.978 (-0.883)
			0.929 (0.960)	0.933 (0.962)	0.315 (0.461)	0.611 (0.738)	0.465 (0.949)	0.742 (0.928)	-0.963 (-0.995)	-0.863 (-0.837)
			0.916 (0.962)	0.926 (0.967)	0.343 (0.486)	0.748 (0.748)	0.604 (0.948)	0.737 (0.932)	-0.960 (-0.989)	-0.791 (-0.849)

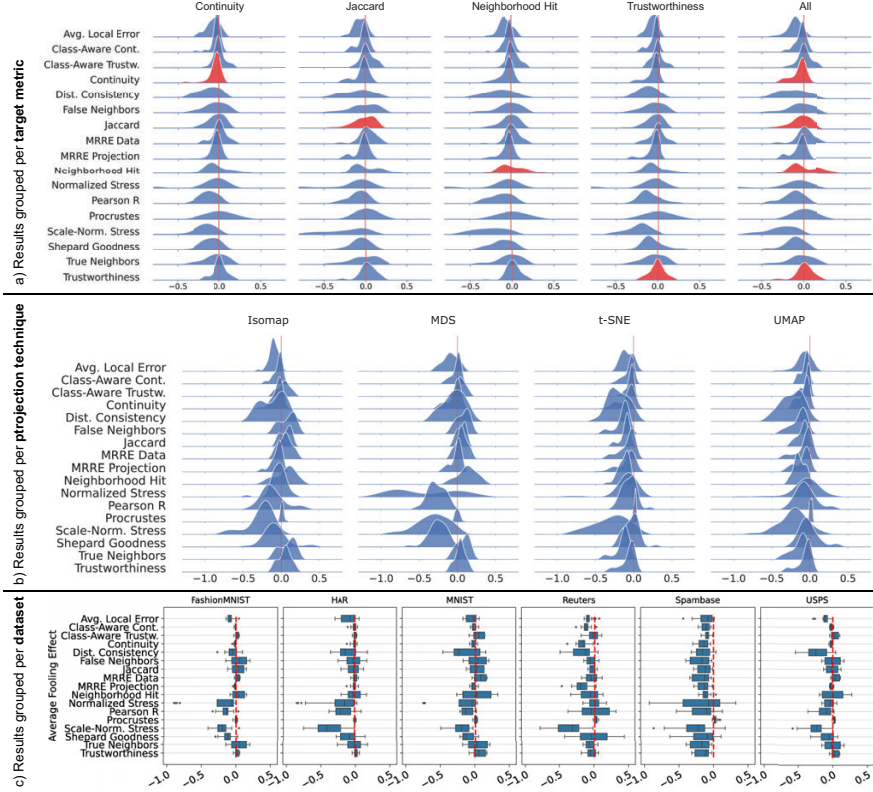


Figure 5.11: Effect of fooling (i.e. distribution of ΔM) grouped (see column headers) per target metric (a), per reference projection (b), and per dataset (c). Metrics are remapped (higher = better). In (a), metrics used as fooling target are highlighted.

5.4.5 Defining a set of guardrail metrics for projection quality

Our experiments showed that (a) quality metrics score high on projections with random or misleading visual patterns and that (b) we can automatically create such projections. So, how can we *still* use metrics to best gauge projection quality? We saw that not all metrics can be fooled equally well — we can, e.g., often deceive Trustworthiness and Neighborhood Hit but not Shepard Goodness (Fig. 5.11 and related text). We also saw that many metrics are correlated, both in their values for different projections (Fig. 5.10a) as well as in their increase or decrease due to fooling (Fig. 5.10b). Metrics that are highly correlated in both these figures measure, to a degree, the same thing. This motivates us to select a small set of *diverse* quality metrics that cover different aspects of a projection. Scoring well in such metrics should be a stronger indication of projection quality. To create this

set, we transform the correlation matrix $\mathbf{C} \in [-1, 1]^{17 \times 17}$ of metric values (Fig. 5.10a) into a distance matrix $\mathbf{D} = \frac{1}{2}(\mathbf{1} - \mathbf{C})$, where $\mathbf{1}$ is a matrix where all entries are 1, such that low distances correspond to highly correlated metrics. We then run HDBSCAN (Campello et al., 2013) with default parameters using $\mathbf{D}$ as input. This produces four clusters of strongly correlated metrics (Distance Consistency, Neighborhood Hit, Normalized Stress, and Procrustes Statistic are not assigned to clusters):

1. Jaccard, False Neighbors, True Neighbors;
2. Avg. Local Error, Pearson r of Distances, Shepard Goodness, and Scale-Normalized Stress;
3. Class-Aware Continuity, Continuity, and MRRE Projection;
4. Class-Aware Trustworthiness, Trustworthiness, and MRRE Data.

Metrics in the same (resp. different) clusters capture similar (resp. different) quality aspects. We pick one metric per cluster to capture different quality aspects — specifically, True Neighbors, Scale-Normalized Stress, Continuity, and Trustworthiness, given their computing simplicity and/or wide adoption in projection evaluation; to these, we add the four metrics which could not be clustered. We next explore how this set of eight ‘guardrail’ metrics can reliably judge projection quality. Table 10 compares the pre- and post-fooling values of these metrics, highlighting the two largest losses in each row. Metrics do not operate on the same scale — smaller losses for one metric may ‘mean more’ than larger losses in other metrics. We see that *no* single metric reliably identifies that a projection is of low quality by returning a low score. In fact, all metrics can even *improve* after our fooling. The metrics showing more dramatic reductions — signaling fooling — are **Distance Consistency** (DSC), **Neighborhood Hit**, the **Pearson correlation r** , and the **True Neighbors** rate. We recommend — within the limitations of our fooling framework — the adoption of this set of quality metrics as a means to increase the reliability of data pattern preservation judgment. Although Stress also signals quality reduction, we do not propose its use due to its unboundedness and scale-sensitivity (Smelser et al., 2024).

5.4.6 Performance

Generating a fooling projection for a dataset from a given projection P with a target metric $\mathcal{M}$ takes 3-4 minutes on average on a commodity PC. This includes training Q_θ , P_ϕ , and $\mathcal{I}_\psi$, running all three post-processing algorithms, and computing the 17 quality metrics for all generated projections. Figure 5.12 compares our PQM computing costs with ZADU (Jeon et al., 2023), to our knowledge the best effort to provide a comprehensive

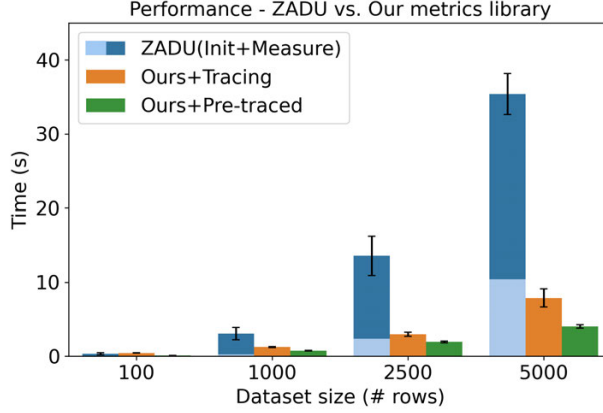


Figure 5.12: Performance of our metrics library *vs* ZADU (Jeon et al., 2023) on 12 metrics common to both. We are 30% slower for datasets of 100 samples due to Tensorflow’s overhead, but see speedups of 2.5-4.5 $\times$ for datasets with > 1K rows including Tensorflow graph creation cost (‘tracing’). Larger speedups (2.8-8.8 $\times$) occur when retracing is not needed. Averaged over 7 runs, error bars show $\pm 1\sigma$.

implementation of PQMs. Our library (Sec. 5.4.1) is 3 to 9 times faster than ZADU.

5.5 DISCUSSION

Likelihood: While we can automatically fool PQMs, one may wonder how likely that is to be done in practice. The chances are slim, but not negligible: Most projection algorithms must be initialized to a setting $\mathbf{Y}_0$. Say we initialize t-SNE to a $\mathbf{Y}_0$ that has been optimized to fool Trustworthiness/Continuity — measures of neighborhood preservation, which t-SNE is designed to do well. Such an initialization could make t-SNE get stuck in that local minimum of good neighborhood preservation, but poor visual quality.

Fooling effect: Our fooled projections have a very different visual signature than the reference ones, yet high PQM values. While arguably such projections can lead users to different conclusions about the depicted data than the reference ones, we aim to test this. Since testing this by user experiments is difficult and of limited coverage (in terms of datasets, projections, tasks to be solved, trained users to recruit as participants), we compare our fooled projections with the reference ones by the Clustering Internal Validation Measure (Liu et al., 2010) VQM with the Silhouette coefficient. This VQM measures aspects such as compactness and cluster separation.

Figures 5.9 and 5.14 show the results with the QM and VQM metrics displayed as annotations. Our fooled projections (both before and after post-processing) have very similar QM values; in contrast, VQM values are very different. This indicates that our fooled projections would convey *different* insights to their users as compared to the reference ones (up to what each VQM measures).

We illustrate this in Fig. 5.13 where we propose a simple task: assess which class has the most *similar* elements — which *visually* corresponds to finding the most tightly-packed cluster. Each projection — MDS and a fooling thereof — shows a different most-dense cluster, illustrating how a user might be led to *distinct* conclusions about the *same* data.



Figure 5.13: Two projections of MNIST (left: MDS; right: our fooling projection for that input). For the task of finding which cluster contains the most similar data points, the two images give different answers — left, the orange cluster (top-middle); right, the dark blue cluster (bottom-right). Hence, our fooling may lead users to different conclusions about the same data. The bar plot to the right shows variation in PQMs, most showing *improvements* in the fooled projection; the largest loss is for the Stress metric.

Limitations: While our results show that one can create projections that are unfaithful to the data while scoring high quality metrics, our findings have some limitations. Firstly, our learning approach has issues handling datasets with only tens of dimensions. We believe this is due to our fixed architecture and may be alleviated by adapting hidden layer sizes when dealing with low-dimensional datasets. Secondly, there is a key mismatch between the assumptions of the deep learning method we use and the semantics of quality metrics. PQMs — even local ones — need access to the *entire* dataset $\mathbf{X}$ and its projection $\mathbf{Y}$ for their computation, since the (point-wise) quality value depends not only on the point itself but on (many) other data points. Yet, in our neural networks, each data point $\mathbf{x}_i$ together with its projection $\mathbf{y}_i$ flows through the network *independently* of all other data points. This trait of Multi-Layer Perceptrons makes them unable to truly

approximate a quality metric. While our postprocessing partially fixes this, the problem remains.

We show experimentally that our suggested set of ‘guardrail’ metrics provides more ‘coverage’ of assessing projection quality. Yet, this conclusion is *not final*: There is a need to (1) develop further metrics that strongly correlate with the human understanding of projections by, for example, combining VQMs based on Abbas et al. (2021); Aupetit and Sedlmair (2016); Jeon et al. (2024b) (among others) with data-driven metrics like Silhouette coefficient (Jeon et al., 2024a; Blasilli et al., 2024); (2) see how well such metrics can resist our fooling (or similar attempts at that); and (3) develop new DR techniques that score high on such metrics, extending the approach of Wang et al. (Wang et al., 2018) beyond linear projections.

5.6 CONCLUSIONS AND FUTURE WORK

We presented a method to automatically create projections of high-dimensional data that exhibit data-unrelated visual patterns — both random and structured ones — which score well on 17 standard quality metrics (PQMs). This provides evidence of the limitations of PQMs in assessing projection quality. We also show that 9 of the studied metrics get fooled even when they are not explicitly targeted by our fooling pipeline. Hence, we claim that the remaining 8 metrics, which are not implicitly fooled, are a good subset to consider in projection quality evaluation.

Our work shows the need of better ways to adequately measure whether/which data patterns a projection preserves. We see several avenues for future work. Our pipeline relies on the model Q_θ which cannot truly approximate a PQM due to its architecture (Sec. 5.5). Using architectures where each $(\mathbf{x}_i, \mathbf{y}_i)$ tuple can integrate information on the projection of its neighbors could make Q_θ better approximate a given $\mathcal{M}$. Transformers (Vaswani et al., 2017) or Nearest-Neighbor Networks (Plötz and Roth, 2018) are potential solutions for this problem. Separately, the ‘strict’ faithfulness of the projection to the data is not always the main goal. Examples include very high-dimensional datasets where volumes of clusters become small — due to the curse of dimensionality — and points tend to lie closer to boundaries; showing such data in a 2D projection would yield highly-clumped data in very thin clusters, something hardly informative. How to measure what a ‘good’ projection in such cases is open for exploration.

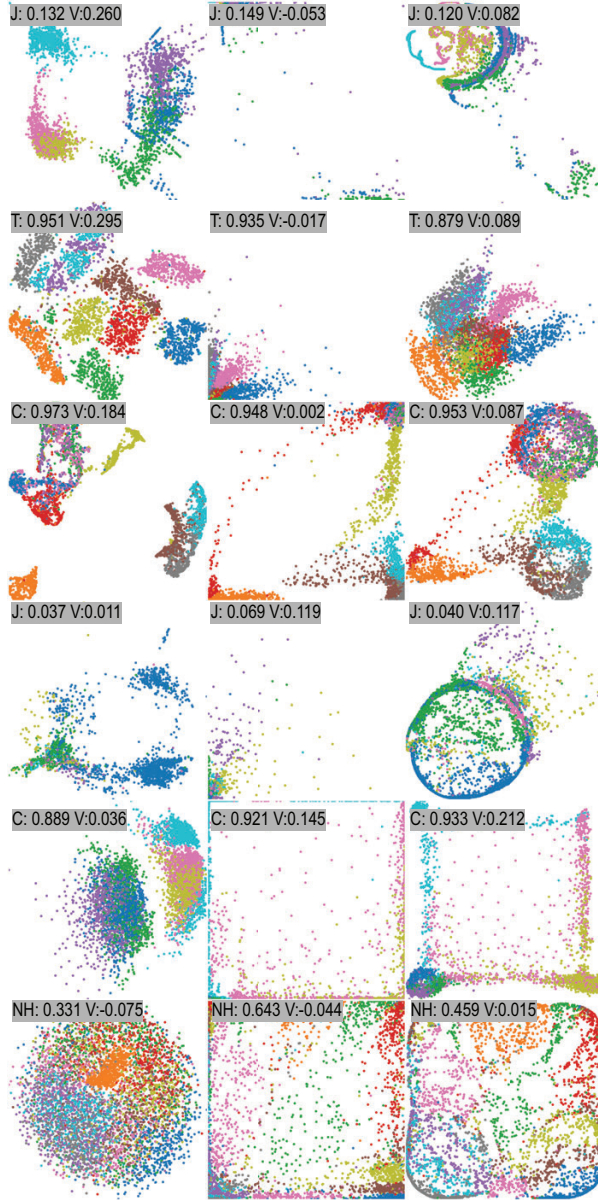


Figure 5.14: Behavior of PQM and VQM (CIVM, [Liu et al., 2010](#)) for reference projections (left column), fooled projections (middle) and postprocessed outputs (right). Annotations show the PQM used to create the fooled projection and the value V of the CIVM. The top three rows show relatively stable PQM values, with clear losses in CIVM. In the bottom 3 rows, PQMs are stable or even increase, and V increases after the destructive and constructive phases, so the arbitrary patterns are not detected by either PQM or CIVM.

UNDERSTANDING CLASSIFIERS WITH ENHANCED DECISION MAPS

Não há incompatibilidade
nenhuma entre a rigorosidade na
busca da compreensão e do
conhecimento do mundo e a beleza
da forma na expressão dos
achados.[†]

Pedagogia da Esperança
PAULO FREIRE

6.1 INTRODUCTION

In Chapters 3 to 5 of this thesis, we have studied techniques for projecting high-dimensional data with the aim of creating interpretable low-dimensional representations that analysts can use to make sense of patterns present in such data. These techniques can be used to study any such high-dimensional dataset. In this chapter, we zoom in on the usage of such techniques for the analysis of a more specific type of high-dimensional artifact, namely a trained *classification model*¹.

Decision Boundary Maps, introduced in Sec. 2.6, are a family of techniques that aim to depict the behavior of such classification models — functions f with discrete outputs — across large portions of the data space. A DBM provides a dense map-like view of a classifier’s behavior by mapping points $\mathbf{p} \in \mathbb{R}^2$ back to the data space $\mathbb{R}^n$ — a process called inverse projection — and next applying f on the inverse-projected data. DBMs generate images similar to the ones in Fig. 6.1 — a running example we refer to throughout the text. These images convey typically the “arg max” output of the classifier (color-coded by class in Fig. 6.1, left); optionally, they can also display confidence information (encoded in luminance in Fig. 6.1, right). While DBMs are far more detailed than aggregated performance statistics, we argue that, without additional information, such maps can risk misinterpretation for the following reasons:

¹ This chapter is based on the paper “Exploring Classifiers with Differentiable Decision Boundary Maps” (Machado et al., 2024d).

[†] “There is not the least incompatibility between rigor in the quest for an understanding and knowledge of the world, and beauty of form in the expression of what is found in that world.”

- Even in regions where the DBM shows the classifier is highly confident, it does not show whether f is classifying data too far removed from any training data point, meaning that the DBM is *wrongly* confident;
- Different regions in the DBM appear qualitatively similar, even when confidence is encoded in the map. Yet, in these regions, the classifier and inverse projection might behave quite differently;
- 2D distances in a DBM do not accurately represent data-space distances since inverse projections are typically nonlinear;
- Neighborhood relations in 2D might not neatly correspond to neighborhoods in data space.

In summary, we argue that the DBM, although useful, *masks* aspects of the high-dimensional and nonlinear nature of both inverse projection and classifier that are interesting for ML practitioners. We address this by *augmenting* DBMs with visualizations that let analysts further answer the following questions:

- Q1:** In which regions of the DBM is the confidence of the classifier f *misleading*?
- Q2:** Where is the model poorly *supported*?
- Q3:** Where is the DBM *distorting* data space distances?
- Q4:** Where is the training of f *sensitive*?
- Q5:** Why is f 's confidence low/high in specific regions of space?

These questions can be phrased in terms of specific types of sensitivity analysis. In other words, we can think about them as asking “what if this aspect were to be perturbed?” It is not surprising then that the tool of choice for this problem is the *derivative*. Differentiation gives us precisely that: the amount of change in outcome effected by a small perturbation in input. The majority of the interactive views that we next propose in Sec. 6.3 are created in this fashion, leading us to term them **Differentiable Decision Boundary Maps** (∂ DBM).

The main contributions of this chapter are as follows:

- ∂ DBMs, a novel set of interactive views aimed at improving the usability of DBMs, relying on differentiable f and P^{-1} .
- A batch-wise implementation of the adversarial example generation algorithm DeepFool (see Section 6.3.2), allowing its concurrent application to thousands of data points.

- An algorithm for deriving an approximate *direct* projection from P^{-1} without any extra training, described in Section 6.4.1.
- The demonstration of the potential of these new tools in an end-to-end dataset poisoning use case.

We implement ∂ DBMs in a Python tool and make our code publicly available at <https://git.science.uu.nl/vig/adversarial-dbm-tool>. The development of the ∂ DBMs provides an answer to our RQ₄.



Figure 6.1: Left: Plain DBM for a Neural Network classifier over the MNIST dataset, generated with NNInv (from a t-SNE projection). Each pixel $\mathbf{p}$ is colored according to $f(\mathbf{p}^\dagger)$. Each resulting zone is annotated with its respective class. Right: the same DBM augmented with classifier confidence encoded in luminance. This is our running example for the rest of the text. In Section 6.3 we show how our ∂ DBMs augment this shallow visualization with additional information.

6.2 PRELIMINARIES

As in earlier chapters, we start by summarizing notation used next; for a full overview, we refer to Sec. 2.2. Let $\mathcal{D} = \{\mathbf{x}_i\}_{i=1,\dots,m}$ be a dataset of samples $\mathbf{x}_i \in \mathbb{R}^n$. When available, a training sample's label y_i can be queried through the function $Y : \mathcal{D} \rightarrow \mathcal{C}$, with $|\mathcal{C}| = K$ classes.

We denote a classifier by $f_\theta : \mathbb{R}^n \rightarrow \mathcal{C}$ (for details, see Sec. 2.7.1). A projection $P_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^q$ maps a high-dimensional point in $\mathbb{R}^n$ to a low-dimensional point; in this work, we consider $q = 2$; for details, see Sec. 2.3. An inverse projection $P_\theta^{-1} : \mathbb{R}^q \rightarrow \mathbb{R}^n$ performs the opposite mapping (see further Sec. 2.6). We will use $\mathbf{p}, \mathbf{p}_0$ to refer to pixels in a DBM, and $\mathbf{p}^\dagger$ as shorthand for $P^{-1}(\mathbf{p})$. The parameters θ are omitted in our discussion

when acceptable. We represent derivatives, gradients, and Jacobian matrices by abusing the $\frac{\partial}{\partial}$ notation as

$$\mathcal{J}P^{-1} \doteq \begin{bmatrix} (\nabla_{\mathbf{p}} P_1^{-1})^T \\ (\nabla_{\mathbf{p}} P_2^{-1})^T \\ \vdots \\ (\nabla_{\mathbf{p}} P_n^{-1})^T \end{bmatrix} = \frac{\partial P^{-1}}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial P_1^{-1}}{\partial \mathbf{p}} \\ \frac{\partial P_2^{-1}}{\partial \mathbf{p}} \\ \vdots \\ \frac{\partial P_n^{-1}}{\partial \mathbf{p}} \end{bmatrix} = \begin{bmatrix} \frac{\partial P_1^{-1}}{\partial p_1} & \frac{\partial P_1^{-1}}{\partial p_2} \\ \frac{\partial P_2^{-1}}{\partial p_1} & \frac{\partial P_2^{-1}}{\partial p_2} \\ \vdots & \vdots \\ \frac{\partial P_n^{-1}}{\partial p_1} & \frac{\partial P_n^{-1}}{\partial p_2} \end{bmatrix}.$$

We will also refer to the Euclidean norm of a vector $\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$ and the Frobenius norm of a matrix $\|\mathbf{A}\|_{\text{Fro}} = \sqrt{\sum_{i,j} a_{ij}^2}$. Finally, a decision map (DBM) is an image where each pixel $\mathbf{p}$ is colored to map the decision, and potentially decision confidence, of a trained classification model f , computed via $f(P^{-1}(\mathbf{p}))$. Using different combinations of direct and inverse projections P and P^{-1} yields different such decision maps, as detailed in Sec. 2.6.

6.2.1 Gradient-based explanations for classifiers

Developing algorithms and techniques to explain the behavior of classifiers is an active area of research (Alicioglu and Sun, 2022; Rosa et al., 2023; Angelov et al., 2021; Samek et al., 2019). This applies especially to so-called Deep Learning techniques which are notorious for their “black-box” aspect: the models thus learned consist of sets of opaque numbers arranged in vectors and matrices and combined through special operations. When deploying such models, practitioners and stakeholders alike can benefit from ways to *probe* it, shining light into their decision-making process, as a way to mitigate risk and assess quality.

Out of all the possible ways to develop such techniques, we devote special attention to those using derivatives (*e.g.*, gradients, Jacobian matrices) to generate explanations for classifications.

In the context of Convolutional Neural Networks (CNNs), which are models of choice when dealing with *e.g.* image data, many techniques focus on providing explanations directly relatable to the data space. Chatopadhyay et al. (2018) proposed Grad-CAM++, where they use gradients to derive visualizations of where the “gaze” of the classifier is directed to which are then overlaid atop the classified image. Similarly, Pan et al. (2021) create maps of feature activations — *i.e.*, which pixels of an image are important for a classification decision — by integrating gradients along paths starting from adversarial examples. Such approaches aim to add explanations that relate directly to the data *points*.

In contrast to the above, we argue for the utility of explanations ranging over a large portion of the data *space*. For this, we extend decision maps,

computed using the NNInv technique (Espadoto et al., 2019b), which we choose due to its genericity and differentiability, to include information provided by gradient-based approaches computed at every map pixel.

6.2.2 Limitations of decision maps

Before further proceeding, it is important to analyze the various sources of inaccuracies (or errors) that influence the computation and interpretation of DBM techniques. We do this in the following in a generic way — that is, without focusing on a particular type of DBM. Putting it differently, the issues outlined below influence *all* existing DBM techniques we are aware of. Along this analysis, we also outline which of such issues our DBM enhancements aim to alleviate.

Direct and inverse projection: The insights obtained from a DBM depend, obviously, on the errors that P and P^{-1} potentially introduce. For P , quality can be measured using the various metrics described in Sec. 2.4. For P^{-1} , quality is typically measured using mean squared error or mean absolute error as a part of the actual optimization process that computes this function (Sec. 2.6). A DBM itself cannot alleviate such errors as it depends on the specific choices made by its designers when choosing appropriate P and P^{-1} techniques. In our work, we propose a way to alleviate such errors by computing P implicitly from a given, differentiable, P^{-1} — thereby minimizing the overall introduced errors (see next Sec. 6.4.1).

Coverage: Recently, Wang and Telea (2024) have shown that all existing DBM techniques cover only a surface-like structure embedded in data space. This fundamentally limits *how much* of the behavior of a classification model can be visualized by a single such map. In the same time, this limits *where* precisely this behavior is visualized (sampled) by the actual setting of this surface in data space — a choice that is made implicitly, and not under user control, by the inverse projection P^{-1} being used. Our work does not alleviate this structural limitation of DBMs.

Information: As already explained, current DBM techniques limit themselves in depicting maximally *two* aspects per map pixel — namely, the classifier’s decision and its confidence. We alleviate this aspect by visualizing five additional aspects per pixel, namely the distance to decision boundaries (Sec. 6.3.2), distance to training data (Sec. 6.3.3), projected space expansion (Sec. 6.3.4), sensitivity to mislabeled samples (Sec. 6.3.5), and class variability in the map space (Sec. 6.3.6). The joint study of all these aspects enriches the analysis of the DBM and reduces potential misinterpretations.

Stability: A separate challenge involves the *stability* of direct (and inverse) projections. Simply put, if small changes in a dataset yield large changes

in P and/or P^{-1} , then both these mappings — and the corresponding DBMs computed using them — are prone to misinterpretation since the details they show may be artifacts of some small-scale data noise. Stability metrics have been proposed to analyze the behavior of projections of time-dependent data (Vernier et al., 2020a), respectively the robustness *vs* different noise types for a single deep-learning-based projection technique, namely NNP (Bredius et al., 2022). However, such analyses have not been used to help the interpretation of DBMs. Oliveira et al. (2023b) have studied the stability of SDBM by visualizing its changes subject to different noise types (similar to Bredius et al., 2022). However, their approach did not consider other P or P^{-1} techniques besides SDBM. More importantly, such visualizations consider artificial data changes (noise). In our work, we consider (and visualize) a more realistic scenario, namely how sensitive a classifier is to mislabeled samples.

6.3 DIFFERENTIABLE DECISION BOUNDARY MAPS

We claim that a deeper understanding of the data and classifier can be obtained by augmenting existing DBMs with additional visualizations. We propose and argue for five such views, most of them obtained by using gradient-based methods. The differentiability of both classifier and inverse projection is crucial in our work, hence we call our proposed views *Differentiable* Decision Boundary Maps. These views can be directly overlaid atop a given DBM and also combined with each other to answer Q1-Q5 from Sec. 6.1. We included numerical scales, aside the color-mapped images, for views where reasoning about the *absolute* values is important; for views which support reasoning about *relative* values, we omit such scales. Our tool always provides tooltips to investigate the exact values at any pixel in any view.

6.3.1 Classifier confidence

As already explained, a plain DBM can be overlaid with the classifier confidence — understood as either the probability assigned to the predicted class or the entropy of the probability distribution over classes — typically encoded into brightness or saturation. Figure 6.2 shows the confidence of the classifier whose DBM was already shown in Fig. 6.1 using color coding. If we are to trust this figure, the classifier appears extremely confident for the vast majority of the points in all decision zones (yellow) except in the direct vicinity of the decision boundaries (green bands). As we shall see, using our additional views tells a different story.

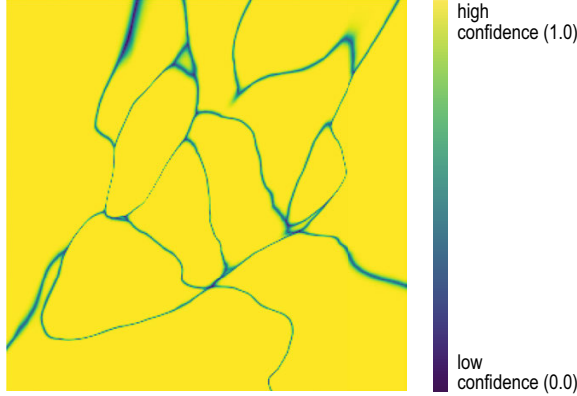


Figure 6.2: Each pixel $\mathbf{p}$ shows the probability assigned to the best class for $\mathbf{p}^\dagger$. We see a classifier that seems uniformly confident of its predictions throughout the represented space.

6.3.2 Distance to decision boundary

Our first DBM enhancement is to show, for each pixel, the distance of the sample $\mathbf{p}^\dagger$ it represents to the actual classifier decision boundaries in data space. This, combined with the confidence map (Fig. 6.2) aims to answer Q1. We compute this distance as

$$\text{Dist}_{\text{Adv}}(\mathbf{p}) = \min_{\Delta \mathbf{x} \in \mathbb{R}^n} \{ \|\Delta \mathbf{x}\|_2 \mid f(\mathbf{p}^\dagger + \Delta \mathbf{x}) \neq f(\mathbf{p}^\dagger) \}. \quad (6.1)$$

Computing Dist_{Adv} exactly is complex. However, techniques for Adversarial Example Generation can approximate it well. In our work, we use DeepFool (Moosavi-Dezfooli et al., 2016) for this goal. Alternative approaches to compute this distance have been proposed in Espadoto et al. (2023a) and Rodrigues et al. (2019). However, these approaches use iterative numerical approximation techniques to search for the closest decision boundary point in the data space, which are sensitive to parameter setting and very slow. DeepFool runs considerably faster and is a more principled approach for the same goal. We also extend the original DeepFool implementation to work on batches of data, concurrently generating adversarial examples for thousands of data points at a time. Overall, we can compute maps of 300^2 pixels in less than a second on a commodity PC (for more detailed performance data, see Appendix A.4.5).

Figure 6.3a shows the distance to decision boundary computed by Eqn. 6.1 for our running example with luminance encoding. Dark areas correspond to low distances; bright areas indicate points that are far away from the decision boundaries. This map focuses attention on points far away from the boundaries. Alternatively, to focus on points close to the

boundaries, which have a higher likelihood to be misclassified upon small data changes or classifier hyperparameter tuning, we can use an inverse luminance mapping (Fig. 6.3b). Pixels close to boundaries are now bright. Comparing these images with Fig. 6.1 or the separately-visualized confidence in Fig. 6.2, it becomes clear that the distance from a pixel $\mathbf{p}$ to the closest boundary in the DBM, *i.e.*, closest DBM pixel of a different color than $\mathbf{p}$, is *not* reflective of the actual distance between the data point $\mathbf{p}^\dagger$ and its closest decision boundary *in data space*. Our maps in Fig. 6.3 show that the data-space distance varies within each DBM region in complex ways. Moreover, while the confidence visualization (Fig. 6.2) suggests that all pixels away from the decision boundaries are qualitatively equal (high classifier-assigned confidence), our distance-to-boundary visualization (Fig. 6.3a) exposes hidden qualitative differences. Some entire regions in the map appear to be completely ‘brittle’ in the sense that they have quite low distances to decision boundaries in data space (see circled region in Fig. 6.3b).

These observations immediately lead us to test what would happen if we introduced mislabeled data points in such fragile regions. For this, we select 10 points $\mathbf{p}_i$ in a given fragile — *i.e.*, low Dist_{Adv} — region, assign wrong labels to them, and add their inverse-projected $\mathbf{p}_i^\dagger$ mislabeled samples to the original training set of our classifier, which contains 5 thousand data points. We keep the number of mislabeled points low to avoid introducing class imbalance to the data. Figures 6.4(b,c) and (d,e) show two such experiments, each using another fragile region. In both cases, we see that it is *easy* to visibly modify the classifier’s decision zones and, thus, behavior. Hence, visualizing such brittle regions is useful to inform classifier designers of areas in data space where the classifier is easily influenced.

6.3.3 Distance to training data

Any classifier is subject to generalization problems (Zhou et al., 2023) — the further a sample is from the training set, the higher is the likelihood that that sample will be misclassified. This is fundamentally a question of *support* (Q2) that current DBMs do not depict. We address this by visualizing the actual distance from the sample corresponding to each DBM pixel $\mathbf{p}$ and the closest point in the training set $\mathcal{D}_t$ as

$$\text{Dist}_{\text{train}}(\mathbf{p}) = \min_{\mathbf{x} \in \mathcal{D}_t} \|\mathbf{p}^\dagger - \mathbf{x}\|_2. \quad (6.2)$$

Figure 6.5a shows $\text{Dist}_{\text{train}}$ for our running example encoded on a blue-to-yellow colormap (blue=small, yellow=large, distances). This view exposes regions of the DBM where, although the classifier might have high confidence values (see Fig. 6.2), it is operating on samples quite different from those on which it was trained. Especially salient is the dark blue

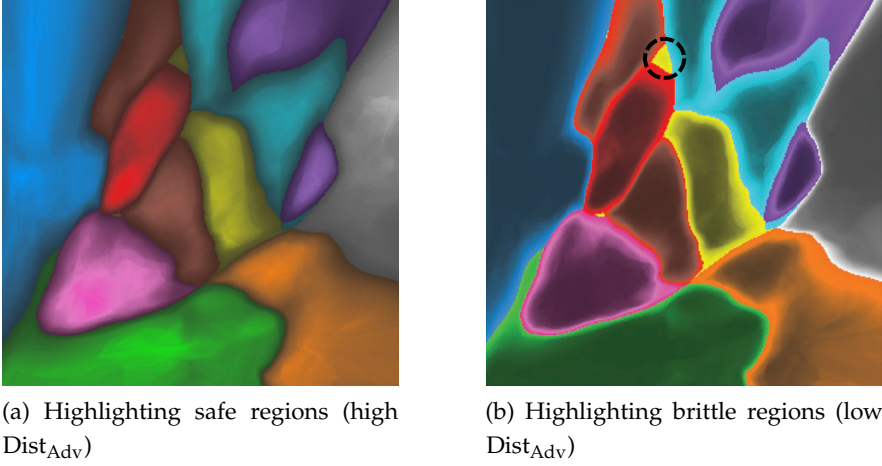


Figure 6.3: Visualizations of approximate distance to the closest decision boundary in data space. Highlights (bright areas) show points that are (a) far away from, respectively (b) close to boundaries. Dist_{Adv} values are scaled to $[0, 1]$ before visual encoding.

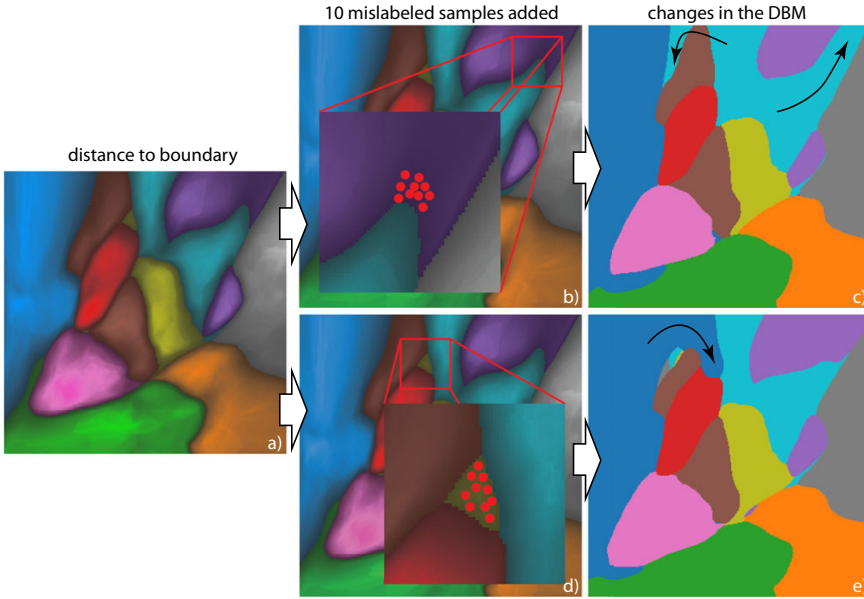


Figure 6.4: Finding brittle regions with low Dist_{Adv} values (a). Adding mislabeled data points in two such regions (b,d) has a strong effect on the DBM. In case (b), the cyan region, class 9, expands so that it fully separates the purple and gray top-right regions (c). In case (d), the blue region of class 0 wraps around to form a single connected decision zone with the new data points (e).

area (Fig. 6.5a, bottom right): the values of $\text{Dist}_{\text{train}}$ are consistently low for that region which corresponds to class 1 (orange in Fig. 6.1 left). This is because all pixels $\mathbf{p}$ in that region back-project close to the training set, meaning that P^{-1} “captures” that region well. Class 1 represents the digit 1 in MNIST, which is the simplest and least-varying shape in the dataset, which in turn is easy to capture. There is further evidence to the above observations. We see that the training samples do not cover the entire region — there are no white dots to the extreme right of the dark blue zone in Fig. 6.5a. Still, this area is overall dark, meaning that P^{-1} “reaches” close to the training data for the entire region, even from points distant from $P(\mathcal{D}_t)$.

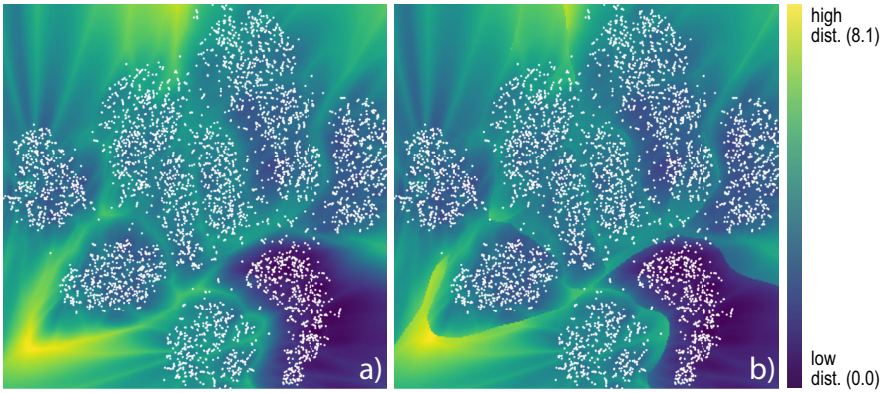


Figure 6.5: Distance to any training point $\text{Dist}_{\text{train}}$ (a) and to training points in the same class as $\mathbf{p}^+$, $\text{Dist}_{\text{SameClass}}$ (b). White points show the training set $\mathcal{D}_t$.

We can be further more specific and ask if a given data region has enough support to be classified as a given class $c \in \mathcal{C}$. We expect good support if a given pixel $\mathbf{p}$ is classified as c *and* is close to some training points of class c in the data space. To check this, for a pixel $\mathbf{p}$ that has label (color) c , we compute the distance of the data point corresponding to $\mathbf{p}$ to the closest training-set point of class c as

$$\text{Dist}_{\text{SameClass}}(\mathbf{p}) = \min_{\mathbf{x} \in \mathcal{D}_t} \{ \|\mathbf{p}^+ - \mathbf{x}\| \mid Y(\mathbf{x}) = f(\mathbf{p}^+) \} \quad (6.3)$$

Figure 6.5b shows $\text{Dist}_{\text{SameClass}}$ for our running example. We see that this distance is subtly different from $\text{Dist}_{\text{train}}$ shown in Fig. 6.5a. Again, the bottom-right region is dark blue, indicating strong support in that region for class 1. The delineation of the dark blue region is now, however, very sharp, matching very well the shape of the orange decision zone (class 1) in Fig. 6.1 left. We also see a clearly delineated yellow region bottom-left in Fig. 6.5b. This tells that the respective region, corresponding to the

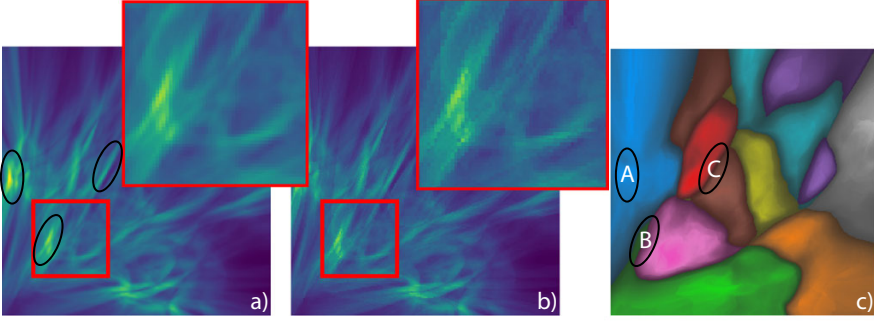


Figure 6.6: Space expansion of P^{-1} visualized by (a) Gradient Maps (Espadoto et al., 2023a), respectively our method (b). Our technique shows more sharp details than Gradient Maps.

border of classes 2 and 6 (green and pink, Fig. 6.1 left), is likely a too far-away extrapolation of the class-2 training. Again, the plain confidence visualization (Fig. 6.1 right) does not show us such insights.

6.3.4 Projected space expansion

We next propose several views based on metrics computed by differentiating specific components of the DBM generation process. This requires introducing some additional notation. We assume we are dealing with a classifier f that has some *internal* scoring that is then transformed (say, by a softmax function) into a probability distribution over classes.

As mentioned in Sec. 6.2, it is possible that P^{-1} itself introduces artifacts in the DBM. We propose a way to visualize how much P^{-1} expands the 2D space locally to be able to invert P with low error rate, therefore answering Q3. Espadoto et al. (2023a) computed this via an approximate pseudo total derivative called Gradient Maps. In contrast, we use the *exact* derivative obtained through automatic *symbolic* differentiation by

$$\text{SpaceExpansion}(\mathbf{p}_0) = \left\| \frac{\partial P^{-1}}{\partial \mathbf{p}} \Big|_{\mathbf{p}=\mathbf{p}_0} \right\|_{\text{Fro}}, \quad (6.4)$$

where we assume that P^{-1} is differentiable with respect to its input. High values of this metric indicate regions where P^{-1} strongly expands the 2D space — that is, close pixels in the DBM map to far-away points in the data space. These are regions where small-scale details in the DBM, *e.g.* the presence of decision boundaries, are very uncertain or even misleading, since zones lying close to each other in the 2D image are *not* neighbors in the data space.

Figure 6.6 shows this space expansion computed by Gradient Maps (a), respectively our method (b). We note that our method is visibly less

blurry than Gradient Maps since the latter uses finite difference approximations which introduce approximation errors that smooth out finer detail, whereas we compute the exact norm of the Jacobian in Eqn. 6.4. We next outline three regions with high space expansion values (yellow areas inside black ellipses, image (a)). Looking at the actual decision zones (image (c)), we see that region A occurs deep inside the blue zone — hence, the expansion of P^{-1} here is not causing interpretation problems. However, regions B and C occur close to decision boundaries — hence, the exact positions of these boundaries in the DBM can be misleading.

6.3.5 Sensitivity to mislabeled samples

As already mentioned, the nonlinear characteristics of both f and P^{-1} mean that plain DBMs mask the complexity of the decision function and of the data space. We also know that the process of training a classifier can be influenced by several aspects, even under fixed hyperparameters. For instance, the initial weights of a neural network are random; the order in which data samples are processed can affect gradient estimation; and the network itself may have stochastic components such as Dropout layers (Srivastava et al., 2014). With this in mind, a classifier designer may be interested in *how sensitive* (Q4) each decision zone is to perturbations that can be introduced by retraining. One way to measure this is to determine how easy it is to create new decision zones for each possible class. The easier this is, the more sensitive the training process is in that DBM region.

We measure this sensitivity by

$$\text{ClassifSensitivity}(\mathbf{p}, c) = \left\| \frac{\partial h[c]}{\partial \mathbf{x}} \Big|_{\mathbf{x}=\mathbf{p}^+} \right\|_2, \quad (6.5)$$

where $h(\mathbf{x})$ is the pre-softmax activation layer of the classifier f and by $[c]$ we mean the score corresponding to class c . Figure 6.7 shows this metric computed for all 10 classes of our running example. For class 1, we see that the sensitivity is high only inside class 1's decision zone (see the actual decision zones in Fig. 6.1 left). This means that adding samples wrongly labeled as class 1 should have little to no effect on the classifier. Conversely, the sensitivity for class 4 shows high values throughout a vast portion of the map. That is, telling class 4 apart from the others is problematic for this classifier, as it can be easily convinced that other digits are class 4 by retraining with mislabeled data points.

We further show the predictive power of these maps with two scenarios (Fig. 6.8). We start with the same DBM (a). Next, we add only 10 mislabeled samples in a region of high sensitivity (b) — the total training set being of 5K samples. We see that the DBM visibly changes (c). Conversely, we add 100 mislabeled samples to a low sensitivity region (d). The DBM now barely changes (e).

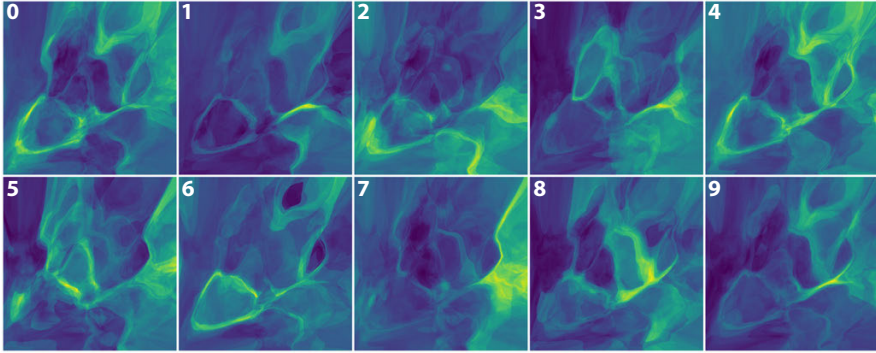


Figure 6.7: Sensitivity metric for each class in the MNIST dataset. Yellow (resp. blue) pixels map through P^{-1} to regions in n-D space where the classifier’s activations change rapidly (resp. slowly).

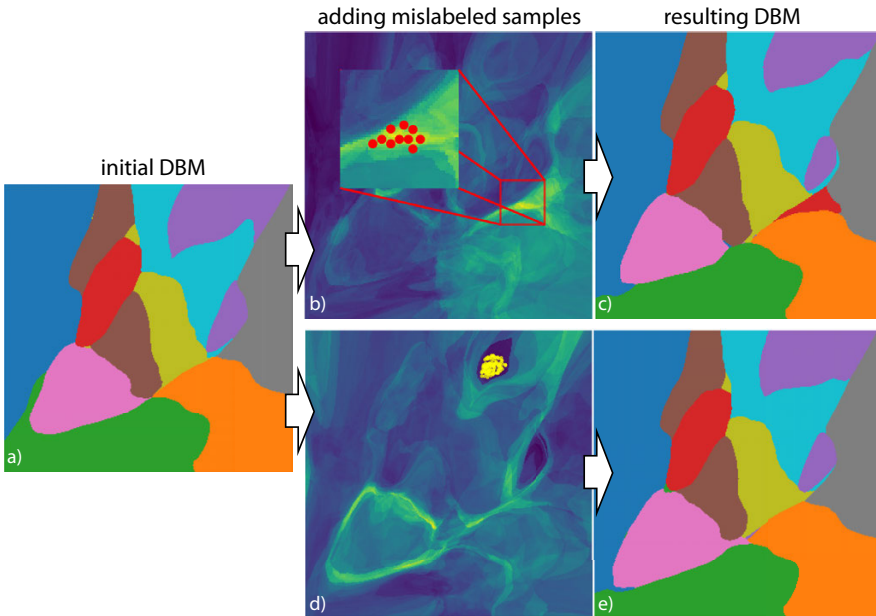


Figure 6.8: Initial DBM (a). Adding mislabeled samples to two regions of high, resp. low sensitivities (b,d) leads to very different, resp. almost unchanged, DBMs (c,e).

6.3.6 Class variability in 2D space

Not all changes in classifier activation patterns result in a different classification output. That is, even if f outputs only one class for a given region of a DBM (a decision zone), it might be the case that f is internally “paying attention” to different aspects of the input. We expect the activation pattern changes to be different from class to class, pointing to different amounts of variability among the elements of each class. This, in turn, can help in judging *why* f ’s confidence varies (or not) in a given region of the DBM (Q5).

Intuitively, we wish to capture the rates of change in the activation patterns $h(\mathbf{p}^\dagger)$ as we move along the DBM. Since we focus on settings where f and P^{-1} are differentiable, we can compute this as

$$\text{ClassVariability}(\mathbf{p}_0) = \left\| \frac{\partial h \circ P^{-1}}{\partial \mathbf{p}} \Big|_{\mathbf{p}=\mathbf{p}_0} \right\|_{\text{Fro}}, \quad (6.6)$$

$$\frac{\partial h \circ P^{-1}}{\partial \mathbf{p}} = \frac{\partial h}{\partial \mathbf{x}} \Big|_{\mathbf{x}=P^{-1}(\mathbf{p})} \frac{\partial P^{-1}}{\partial \mathbf{p}}.$$

Computing ClassVariability for every DBM pixel shows the amount of change expected in the activation patterns in a region of the data space corresponding to an infinitesimal movement in the 2D map. This goes one step further than the Gradient Maps (Espadoto et al., 2023a). Differentiating through the classifier f as well as the inverse projection P^{-1} allows for deeper insights into the inner workings of f .

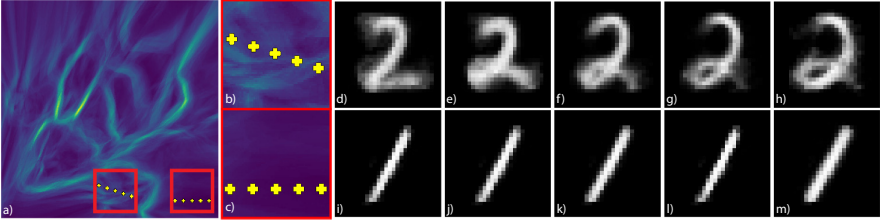


Figure 6.9: Comparing equally-spaced points in the projected space, we notice that in regions where ClassVariability is high (b), the elements obtained through P^{-1} change visibly — be it within a given decision zones or across them (d-h). Conversely, when the metric is low (c), we are in a region where there is less variation in the data (i-m).

Figure 6.9a shows ClassVariability for our running example. Bright yellow regions tell that neighbor map pixels correspond to strong changes of the classifier behavior. Not surprisingly, many such changes happen along *borders* between different regions in the DBM. However, our visualization shows that changes also happen *within* those regions — a fact that the original DBM doesn’t convey. These within-region changes are not large

enough to cause a switch in the classifier’s *prediction* — if they were, they would create new decision zones. Still, they tell that the internal activations of the classifier are changing, pointing to *class variability*. We can see this by sampling points in regions with high values of this metric (Fig. 6.9b) and comparing their variability to points from low value-regions (Fig. 6.9c). For the former case, we see more variability (images d-h) than for the latter case (images i-m). The class variability shows more activity than the classifier confidence visualization (Fig. 6.2) — the latter spikes to 1 practically everywhere except on the decision boundaries. This shows a uniform confidence in the face of non-uniformly-similar samples (Fig. 6.9, d-h).

Summarizing the above observations, Table 11 shows how we can combine the class variability and classifier confidence views to draw several conclusion on a classifier’s behavior.

Table 11: Cross-referencing Class Variability with the Confidence map and conclusions that can be drawn.

		Class Variability	
		Low	High
Confidence	Low	Classifier could not learn the main pattern of this region properly.	Classifier could not learn the patterns of this region properly.
	High	Data pattern is simple enough to be grasped by the classifier.	Data varies considerably, but the classifier can still learn the proper class.

6.4 CASE STUDY: DATA POISONING

We now describe a use case for our DBM augmentations — assessing whether a data poisoning attack is being carried out when updating a classifier. The need to update a classifier arises from time to time and can be due to a multitude of factors (Lu et al., 2019; Ditzler et al., 2015). For instance, the data distribution might have changed — e.g., user preferences tend to change over time —, or a new source of data may need to be added into a system. In this process, the inclusion of new samples opens up the possibility of an attacker crafting specific training data aimed at *harming* the accuracy of the classifier being updated. These malicious actions are called Data Poisoning attacks, and have been studied extensively (Cinà et al., 2023; Cinà et al., 2022). The particular type of attack we focus on falls under the category of so-called *causative attacks* (Barreno et al., 2010) — in short, attacks that are carried out by tampering with training data —, which we next briefly describe following Steinhardt et al. (2017).

The scenario consists of a *defender* learning a model f_θ and an *attacker* who wants the learned f_θ to incur a high test loss (which the defender tries to minimize).

- We establish a “clean” dataset $\mathcal{D}_c$ consisting of n data points.
- The attacker chooses a “poisoned” dataset $\mathcal{D}_p$ of ϵn data points, where $\epsilon \in [0, 1]$ is the attacker’s budget.
- The defender trains on the full dataset $\mathcal{D}_c \cup \mathcal{D}_p$, producing a model $f_{\hat{\theta}}$ and incurring test loss $\mathcal{L}_{\text{test}}(\hat{\theta})$.

We next show that if the attacker chooses points close to decision boundaries in the data space and tampers with their associated labels, they can negatively impact the test loss, fulfilling their goal. Hence, our ∂ DBMs can be used as a guide for an attacker to successfully craft poisoned data. Conversely, this also means that using our DBM visualizations can potentially *uncover* this type of attack before it takes place, enabling the defender to filter out the undesirable data.

6.4.1 Preparation: implicit projection

As already explained, to create our DBM visualizations, we need to be able to run P^{-1} on the projection space. This can be trivially accomplished by using NNInv, as we did for all examples already shown so far. However, to place new data in the ∂ DBM, we need to project *unseen* data points via P . Not many projection algorithms support this *out-of-sample* capability — for example, neither t-SNE nor UMAP do. We could solve this by looking for *parametric* projection algorithms, such as PCA, Parametric t-SNE (van der Maaten, 2009), or Auto-Encoder based methods. This would (unnecessarily) impose a limit to the user of ∂ DBMs, which goes against our goal of ∂ DBMs being as generic as possible.

Another solution would be to use an approximation to the projection, such as NNP (Espadoto et al., 2019a) — a neural approach that generalizes any projection to unseen data. However, NNP has three disadvantages: (1) it does not always perform well in generalizing the projection it is trained on; (2) using it means introducing *yet another* component in our pipeline that must itself be trained and evaluated; (3) there is no guarantee that the NNP-learned approximation of P is compatible with the P^{-1} already present in the ∂ DBM pipeline. That is, the round-trip error $\|P_{\text{NNP}}(P^{-1}(\mathbf{p})) - \mathbf{p}\|_2$ and/or $\|P^{-1}(P_{\text{NNP}}(\mathbf{x})) - \mathbf{x}\|_2$ can be very high.

To address the above issues, without the need for a parametric P , we propose a way to perform a direct projection P *implicitly* from a P^{-1} that is differentiable with respect to its input (a requirement we had earlier for

some of our techniques). For each point $\mathbf{x}_{\text{new}} \in \mathbb{R}^n$ we wish to project, we solve the optimization problem

$$\hat{P}(\mathbf{x}_{\text{new}}) = \operatorname{argmin}_{\mathbf{z}} \|P^{-1}(\mathbf{z}) - \mathbf{x}_{\text{new}}\|_2^2. \quad (6.7)$$

This is easily done with standard optimizers such as Adam (Kingma and Ba, 2015). Since the optimization goal is non-convex because of P^{-1} , we use a few tricks to ensure convergence to a sensible projected point — see Algorithm 1. We perform this optimization batch-wise, which makes it reasonably fast: on a commodity PC, projecting $m = 100$ points takes about 0.3 seconds; for reference, projecting the same points with t-SNE takes equal time.

Algorithm 1 starts by generating κ candidate points for each input data point $\mathbf{x}'_i$ that should be projected. We can either generate those uniformly at random in the projected space (lines 9-11), or use a “warm start”. In the latter case, the candidate set for each $\mathbf{x}'_i$ is generated by first finding the data point $\mathbf{n} \in \mathcal{D}_i$ most similar to $\mathbf{x}'_i$ (line 2), then generating κ points by adding Gaussian noise to $\mathbf{n}$ (line 6, $\sigma = 0.01$ in our tests). The generation of κ candidate points per input point to Algorithm 1 is a way to increase the chances of finding a good minimum to the reconstruction error. These points are generated in 2D, not in n -D, which avoids the curse of dimensionality; still, κ should not be set too high since the complexity of the algorithm depends linearly on it ($\kappa = 10$ produces good results in our tests).

Once we have, for each input data point $\mathbf{x}'_i$, a set of candidate projected points $\mathbf{Z}_i \subset \mathbb{R}^2$, Algorithm 1 proceeds by iteratively optimizing the candidate points $\mathbf{z}_k \in \mathbf{Z}_i$ (lines 13-19) so as to minimize the reconstruction error through P^{-1} . We limit the number of times this optimization is done with the hyperparameter T , fixed to 50 in our tests. Finally, we select the element of each candidate set that achieved the smallest reconstruction error (line 20). These are the implicitly projected versions of the input points.

Figure 6.10 shows our method in use. The left image shows the t-SNE projection P of the MNIST dataset. From this projection, we construct P^{-1} using NNInv, as mentioned earlier. The right image shows the result of $\hat{P}$ (learned from P^{-1}) on unseen data points. As the colors show, the unseen points are projected as expected, given the projection P , within their respective point clusters.

6.4.2 The attack setting

As a dataset, we use ciFAIR (Barz and Denzler, 2020), a variant of the well-known CIFAR dataset with duplicate images removed. We use a pretrained ResNet (He et al., 2016) neural network as a feature extractor, mapping each 32×32 RGB image to a feature vector in $\mathbb{R}^{4096}$. As classifier instance,

Algorithm 1 Implicit Direct Projection ($\hat{P}$)

Require: $P^{-1}, \mathbf{X}_{\text{new}} = \{\mathbf{x}'_i, \dots\}$, WarmStart $\in \{\mathbb{T}, \mathbb{F}\}$, # seeds $\kappa \geq 1$, # iterations $T > 0$.

Ensure: $\mathbf{O} = \{\mathbf{o}_i, \dots\}$, the projected points from $\mathbf{X}_{\text{new}}$.

```

1: if WarmStart then
2:    $\mathbf{N} \leftarrow \{\arg \min_{\mathbf{x} \in \mathcal{D}_i} \|\mathbf{x}'_i - \mathbf{x}\|_2^2 \mid \mathbf{x}'_i \in \mathbf{X}_{\text{new}}\}$ 
3:    $\mathbf{n}_i^{(q)} \leftarrow P(\mathbf{n}_i) \forall \mathbf{n}_i \in \mathbf{N}$ 
4:   for all  $\mathbf{x}'_i \in \mathbf{X}_{\text{new}}$  do
5:      $\triangleright \mathbf{Z}_i$  is the candidate set for point  $\mathbf{x}'_i$ 
6:      $\mathbf{Z}_i \leftarrow \{\mathbf{n}_i^{(q)} + \varepsilon_k \sim \mathcal{N}(0, \sigma^2) \mid k \in \{1, \dots, \kappa\}\}$ 
7:   end for
8: else
9:   for all  $\mathbf{x}'_i \in \mathbf{X}_{\text{new}}$  do
10:     $\mathbf{Z}_i \leftarrow \kappa$  uniformly distributed points in  $[0, 1]^q$ 
11:  end for
12: end if
13: for  $j = 1, \dots, T$  do
14:   for all  $\mathbf{x}'_i \in \mathbf{X}_{\text{new}}$  do  $\triangleright$  implemented batch-wise parallel
15:     for all  $\mathbf{z}_k \in \mathbf{Z}_i$  do
16:        $\mathbf{z}_k \leftarrow \text{OptimizerStep}(\mathbf{z}_k, \nabla_{\mathbf{z}_k} \|P^{-1}(\mathbf{z}_k) - \mathbf{x}'_i\|_2^2)$ 
17:     end for
18:   end for
19: end for
20:  $\mathbf{o}_i \leftarrow \arg \min_{\mathbf{z}_k \in \mathbf{Z}_i} \|P^{-1}(\mathbf{z}_k) - \mathbf{x}'_i\|_2^2 \forall i$   $\triangleright$  best candidates
21: return  $\mathbf{O}$ 
    
```



Figure 6.10: Left: t-SNE projection of the MNIST dataset, colored by ground truth labels. Right: implicit projection $\hat{P}$ of unseen data obtained through Alg. 1.

we next train simple 3-layer feed-forward neural networks that take these 4096-dimensional vectors and aim to predict the correct image class.

To carry out the attack in a way that allows for analysis, we first train a model f^{clean} using $m = 5000$ data points from the training set ($\mathcal{D}_c$). The test accuracy is measured on a held-out set $\mathcal{D}_{\text{test}}$. The attacker then chooses ϵm new data points to form $\mathcal{D}_p$. A new model f^{att} with the same architecture and hyperparameters as f^{clean} is trained on $\mathcal{D}_c \cup \mathcal{D}_p$. The test accuracy of f^{att} is measured on $\mathcal{D}_{\text{test}}$ to demonstrate the effect of the attack.

6.4.3 Carrying out the attack

We next discuss how we generate $\mathcal{D}_p$ and the effect that training on the extended dataset has on test accuracy.

Generating poisoned data. In our simplified scenario, we generate the set of ϵm samples that have not been seen by any of our models by back-projecting (P^{-1}) points that are *close to a decision boundary* (see Section 6.3.2). We choose such data points because, as explained earlier in Sec. 6.3, decision boundaries are the regions where effecting changes on the classifier's outputs should be easiest. Figure 6.11(left) shows the t-SNE projection of the ciFAIR dataset created using t-SNE and, also, the DBM of the trained classifier. The right image shows the selected samples close to decision boundaries.

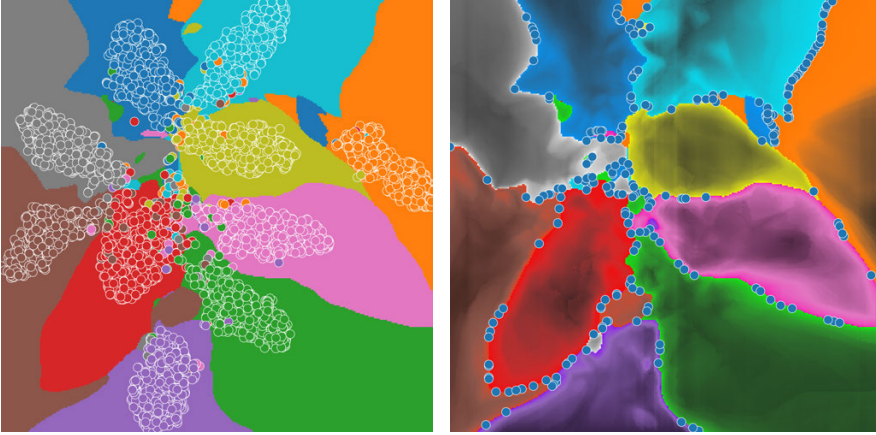


Figure 6.11: Left: the ciFAIR dataset projected with t-SNE. Right: generated poisoned points $\mathcal{D}_p$ using $\epsilon = 0.05$. These all lie close to the decision boundaries of the classifier.

To poison the dataset in a simple way, we add the label 0 to these selected samples. A classifier that is able to predict these *incorrect* labels



Figure 6.12: The DBMs for the ciFAIR dataset before (left) and after (right) the data poisoning attack. We see that clear differences appear, in the form of new decision zones for class 0, but also in the changes in the shapes of other classes’ zones. This reconfiguration of the decision surfaces is directly reflected in the change in test accuracies (Table 12).

should intuitively perform worse on the held-out $\mathcal{D}_{\text{test}}$, since we train it with misleading data.

Table 12 shows the results of the attack for two different ϵ values. We are able to dramatically lower per-class test accuracy by 3.7% (for $\epsilon = 0.02$) and 5.8% ($\epsilon = 0.05$) by this simple attack. The overall test accuracy drops by 1.3% ($\epsilon = 0.02$) and 2.2% ($\epsilon = 0.05$), taking into account the different relative amounts of points from each class.

Interestingly, the accuracy for class 0 — used for our added poisoned points — *rose* for $\epsilon = 0.05$. We explain this by observing that, indeed, some of the new points are similar to true members of class 0, hence introducing a correct signal in training; also, the training process itself is prone to

Table 12: Variation in test accuracy measured on unseen data before poisoning ($\mathcal{D}_c$) and after poisoning with different budgets ($\epsilon = 0.02, 0.05$). We highlight the classes for which we were able to drive down test accuracy.

	Test Accuracy per Class									
	0	1	2	3	4	5	6	7	8	9
$\mathcal{D}_c$	83.1%	90.5%	90.0%	82.0%	86.2%	77.5%	89.9%	93.5%	94.7%	89.4%
$\epsilon = 0.02$	79.4% (-3.7%)	88.1% (-2.4%)	86.8% (-3.2%)	82.5% (+0.5%)	83.8% (-2.4%)	78.5% (+1.0%)	91.0% (+1.1%)	91.5% (-2.0%)	95.2% (+0.5%)	86.9% (-2.5%)
$\epsilon = 0.05$	86.8% (+3.7%)	85.7% (-4.8%)	84.1% (-5.8%)	80.7% (-1.4%)	83.3% (-2.9%)	77.5% (+0.0%)	87.8% (-2.1%)	91.0% (-2.5%)	91.8% (-2.9%)	86.4% (-3.0%)

perturbations, so the classifier may have learned better from the original training samples of class 0 when performing this experiment.

We conclude that ∂ DBMs are simple but effective tools that provide starting guidance to an attacker wishing to negatively impact the test accuracy of a classifier by directly pointing them to regions that reliably affect classification outcomes.

6.4.4 Defending using ∂ DBMs

In turn, a defender can also use ∂ DBMs to thwart a poisoning attack. Say the defender has received a dataset $\mathcal{D}_{\text{mixed}} = \mathcal{D}_{\text{old}} \cup \mathcal{D}_{\text{new}}$, potentially poisoned as described in Sec. 6.4.3. We assume the defender has the needed information to decide whether poisoning is suspected or not, *e.g.*, can tell such attacks apart from concept shifts in the data-generating distribution; this type of decision is out of our scope. The attacker may have crafted points in $\mathcal{D}_{\text{new}}$ in the manner of Fig. 6.11, but the defender has no way of knowing that. The defender can however inspect these new training points by projecting them onto the ∂ DBMs of the existing classifier. This can be directly done using Alg. 1.

The defender can now use ∂ DBM visualizations to explore $\mathcal{D}_{\text{mixed}}$. Figure 6.13(left) shows the dataset projected atop the DBM of the classifier. For clarity, we only show here the $\mathcal{D}_{\text{new}}$ points — these can be easily isolated from $\mathcal{D}_{\text{mixed}}$ since we have $\mathcal{D}_{\text{old}}$. In our image, the new points appear close to the decision boundaries, which raises suspicions. The right image explores further by showing the distance to decision boundaries (Sec. 6.3.2). We now see that all points are very close to the *actual* decision boundaries — even some points which, in the left image, appeared to be further away from the decision boundaries in the *image*. The defender can now conclude that a dataset poisoning attack is underway; the attack can next be thwarted by either filtering out the suspicious samples or choosing not to update the classifier altogether.

6.5 DISCUSSION

We discuss our method along several aspects, as follows.

Simplicity vs interpretability: ∂ DBMs can be created automatically, with zero parameter-setting efforts, for any differentiable classification model, that is, with very little effort even for inexperienced users. Interpreting the set of views that ∂ DBMs provide, however, requires a certain amount of training and effort. We argue that such explanatory views are *necessary* for any DBM algorithm. Indeed, such algorithms (1) use direct and inverse projections, which are subject to errors; and (2) as recently shown by Wang

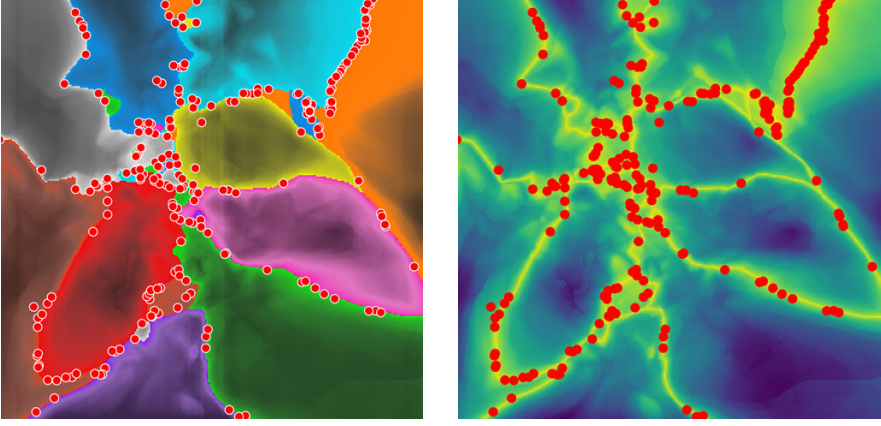


Figure 6.13: A defender needs to project the new dataset using $\hat{P}$. The new data points $\mathcal{D}_{\text{new}}$ are shown here for the same scenario as above atop the distance to decision boundary view. On the left, we embed it on the DBM (Section 6.3.2) and on the right, we display it independently.

et al. (2023); Wang and Telea (2024), all current DBM techniques only visualize a *surface-like* subset of the full data space the classifier works on. Having the *option* to visualize ∂ DBMs, either as separate views, or overlaid atop a plain DBM, aids in answering specific questions (Q1-5) which, as we have shown, cannot be answered by a plain DBM. We argue that this is valuable even for inexperienced users.

DBM distortions: As explained earlier, all current DBM methods have limitations, either in terms of which part of the data space they show, or the distortions they introduce due to the (inverse) projections they use. We do not aim to *correct* these distortions (or limitations), as this would imply fundamentally changing the DBM construction algorithm. Rather, our explanations work generically for any DBM technique which uses a differentiable inverse projection. As such, we focus on *highlighting* the place, nature, and extent of problems caused by DBM distortions, rather than aiming to fix these. Similar approaches are well known for visualizing (and not correcting) distortions caused by direct projections (Heulot et al., 2017; Martins et al., 2014; Aupetit, 2007; Lepinats and Aupetit, 2011).

6.6 CONCLUSION

We have proposed ∂ DBMs, a set of visualizations that, independently or as a whole, enable a deeper analysis of Decision Boundary Map (DBM) depictions of classification models. Our views are mainly based on one simple concept — measuring the sensitivity of different elements of the

DBM pipeline (inverse projection, classifier) with respect to their inputs, and visualizing how this changes across the DBM. Our techniques can be generically applied to any data dimensionality and to DBMs constructed using any direct projection technique. We show the additional information that can be derived from our visualizations, linking them to concrete uses and demonstrating their explanatory power for classification. We also illustrate our ∂ DBMs for the task of creating, but also defending against, an end-to-end dataset poisoning attack. As a side contribution related to this use-case, we also show how to construct an implicit projection function based on a given inverse projection.

∂ DBMs, however, also have some limitations. Our techniques require the differentiability of the inverse projection used by the DBM and the classification model itself. However, most ML models (and importantly, deep neural networks), and most inverse projection techniques we know of, fall in this class. Visualizing the *exact* distance to decision boundaries of a classifier (Section 6.3.2) would be computationally intensive, limiting the interactivity of the created maps. Moreover, such exact maps could be very noisy due to the complex hypersurface determined by a classifier, so approximating that distance both increases speed and provides some regularization to the maps. More importantly, ∂ DBMs do not overcome an important limitation of their predecessors — they heavily rely on the quality of the generated (direct) projection of the data. Poor projections will lead to poor maps. As such, one still requires exploration to choose a suitable projection technique. An alternative to this that is worth studying is using techniques that provide some regularization of the projected space, such as the recently-proposed ShaRP (Machado et al., 2023).

We see several immediate avenues for future work. ∂ DBMs can be extended by additional, more specialized, views that aim to answer more complex, specific, questions. Adding the support for interactive querying of the maps would allow ML engineers to pose more complex queries on parts of the data and/or the map, thereby narrowing down problems of existing classifiers. Separately, recent and parallel developments have shown how DBM computation can be accelerated using coherence analysis of adjacent map pixels (Wang et al., 2025). The respective work, however, did only consider plain DBMs, that is, consisting of per-pixel label and confidence information. It is definitely interesting to consider extending such approaches to all our additional per-pixel attributes that differentiable DBMs consist of. Finally, using ∂ DBMs in a controlled study to demonstrate their effective added-value in an end-to-end ML engineering task is an important step towards practical validation, along directions set by recent work that illustrate the validation of plain DBMs (Benato et al., 2024b).

CONCLUSIONS

To be whole is to be part;
true voyage is return.

The Dispossessed
URSULA K. LE GUIN

We close this thesis by recalling its proposed contributions and critically assessing them, as well as pointing towards directions for future work.

In Chapter 1, we discussed possible interplays between Machine Learning and Dimensionality Reduction: two fields that have, independently and concurrently, enlarged the realm of possibilities when it comes to knowledge acquisition from data. We decide to explore in more depth the synergies of the use of Neural Networks — a Machine Learning technique — for Dimensionality Reduction, leading to the following research questions into what we call *Neural Dimensionality Reduction*:

RQ1: Can we create neural projections having all desirable properties of such state-of-the-art methods while reducing fuzziness?

RQ2: Can we create neural projections having all desirable properties of such state-of-the-art methods and, additionally, offering an explicit control of the point patterns they create?

RQ3: Are high values of projection quality metrics a sufficient indicator that a projection faithfully captures underlying data patterns?

RQ4: How to extend decision boundary map visualizations in order to expose additional insights regarding the operation of the visualized classification model?

We have explored these research questions throughout Chapters 3 to 6. We now collect our conclusions and how they answer each one of our proposed research questions.

We addressed RQ1 in Chapter 3 by proposing a novel algorithm called APPA — the Approximating Parametric Projection Algorithm — based on using a suitable regularization technique in order to reduce the fuzziness problem associated with previous Neural Projection techniques such as NNP or its (slightly) superior versions. Our results demonstrate the success of our approach, showing — both visually and in terms of projection quality metrics — that both APPA and its sampled version SAPPA are effective at faithfully reproducing their training projections significantly alleviating the fuzziness problem. We attain this with *zero* extra cost at inference — *i.e.*, projection — time, since (S)APPA's complexity is limited to the

training loop. This means (S)APPA have all desirable properties of state-of-the-art methods while sidestepping the biggest obstacle to the adoption of, *e.g.*, NNP.

(S)APPA exploits a highly versatile aspect of Neural Networks: Anything that can be differentiated and whose minimization represents some aspect we would like to achieve in the function computed by the network can be used as a loss term. In APPA, the additional loss term is a KDE-based prior probability distribution of points in the projection space. In SAPPA, it is the distance to a sampled barrier function computed also based on a KDE prior.

RQ2 is addressed in Chapter 4 with the introduction of ShaRP — the Shape-Regularized Projection algorithm. Our experiments show that ShaRP’s VAE structure allows it to produce sensible projections of high-dimensional data that score well in terms of projection quality metrics, all the while allowing users to control exactly which visual patterns show up in the obtained projection. While other algorithms display visual patterns as an indirect consequence of their algorithmic design, ShaRP is the first algorithm to allow explicit control thereof. ShaRP’s main drawback is the requirement of *labels* or *pseudolabels* — the latter obtained by clustering — and its reliance on this class information in order to produce visually appealing projections. However, this investigation opens a research direction into what kind of *inductive bias* a Neural Network must have in order to be able to forego label information and still produce high quality projections.

In the context of ShaRP, the synergy between Neural Networks and Dimensionality Reduction is crucial: Without being able to explicitly enforce a probability distribution family over the latent space, such explicit control over visual patterns is, to the best of our knowledge, impossible. The Variational Auto-Encoder is what bridges this gap in Dimensionality Reduction techniques.

To answer RQ3, we develop a so called Projection Quality Metric Fooling pipeline in Chapter 5. We show how it is *easy* to automatically, with no direct human intervention, produce projections that show *no* distinguishable visual patterns coming from the data, yet score high according to PQMs. Despite ranging over a wide variety of datasets, projections, and target metrics, our experiments all point towards the same conclusion: Projection Quality Metrics *are not sufficient* in characterizing a projection as good in terms of data pattern preservation. They are, nevertheless, *necessary*: When a PQM produces a low value, it reflects a projection that surely does *not* preserve data patterns. The evidence collected in this study points to the need for more *complex* PQMs. Complexity here should not be understood as how intricate the computation of a given metric is; rather, robust metrics should mimic more closely how human perception works — *i.e.*, they should operate at many scales at once, being neither global nor neighborhood-based.

Once more, the properties of Neural Networks are essential in developing the Fooling pipeline. Indeed, our objects of interest — Projection Quality Metrics — are typically non-differentiable, involving , *e.g.*, sorting operations. Nevertheless, using NNs for (imperfectly) approximating PQMs is key in allowing our pipeline to morph projections into poor ones, often with little impact to PQM values. That is due to the fact that we can differentiate *through* the PQM approximator, allowing the optimization of the projection itself to be *guided* by the PQM values.

Finally, we reach towards the boundary of the field of Dimensionality Reduction at the point where it touches Explainable AI. In Chapter 4, we give a first contribution to the answer of RQ4. ShaRP, being an auto-encoder based algorithm, contains in itself by design a mechanism to generate Decision Boundary Maps. We explore the differences in how DBMs are generated by ShaRP compared to its predecessor technique, SSNP.

Similarly, in Chapter 3, there is another potential contribution to RQ4. With APPA being a projection algorithm, its output projection can also be used to build DBMs. A comparison between APPA and other techniques, exploring the effect of its regularization on the inverse projection, is a clear direction for future work.

In Chapter 6 we provide a more robust answer to RQ4. We see how Decision Boundary Maps can be *misleading* due to their over-simplification of the phenomenon they depict. Namely, DBMs represent a classifier’s behavior by flat neighboring regions — the Decision Zones — without any other topological information derived from the data space. We show how the addition of gradient-based views — hence the name of our technique, the Differentiable DBM (∂ DBM) — provides information about regions where the visualized classifier is *overconfident*, for example. Not only that, but we visualize regions of the DBM that represent areas close to decision boundaries in the data space. In these regions, the addition of wrong training examples can cause a strong decision shift in the classifier, leading to the incorrect classification of *many* other data points. The explored dataset poisoning scenario further demonstrates this effect.

In building the ∂ DBMs we use Neural Networks to compute inverse projections; we also focus our attention on visualizing differentiable classifiers — often Neural Networks themselves — due to their widespread use and high capacity of computing complex classification functions. This allows the end-to-end differentiation of the inverse projection-classification ensemble, which is our final explored example of the synergy between Neural Networks and Dimensionality Reduction.

As a whole, this thesis explores what we recognize as a field of its own, which we refer to as *Neural Dimensionality Reduction*: the use of Neural Networks in conjunction and cooperation with Dimensionality Reduction techniques. We showed how these two families of techniques go hand-in-hand in providing us novel ways to think about, develop, evaluate, and

explain DR algorithms and ML models. Our results demonstrate the desirable properties enjoyed by Neural Dimensionality Reduction methods: The described algorithms and techniques are fast, highly parallelizable, parametric, generic over datasets and require minimal hyperparameter tuning (in each chapter we use a *single* setting for the hyperparameters across all experiments). Further, they enable capabilities that are not (easily or directly) attainable by ‘classic’ Dimensionality Reduction. We believe to have made a strong contribution to the case for the benefits that can be obtained by investigating and actively exploiting the synergies between Neural Networks (and Machine Learning in general) and Dimensionality Reduction. We next discuss limitations in the work described by this thesis and avenues for future work.

7.1 FUTURE WORK

Looking back at the line of work developed in this thesis, we explore directions for future work.

Robustness of quality metrics: In Chapter 5, we have shown the strong limitations of state-of-the-art Projection Quality Metrics, to wit: They are unable to properly and reliably tell apart good projections from bad ones. Towards the end of that chapter, we propose a set of PQMs that, according to our limited experimentation, might provide a more complete assessment of projections, reducing ‘false positive’ judgments. We see this solution as a palliative, however. There is space for the development of new, more complex Projection Quality Metrics that are able to more closely match the human perception of projections. These metrics should operate on the level in which visual patterns form, relating those to the data used to generate the projection. This could be achieved by an iterative coarsening procedure that measures the degree of pattern preservation at increasing scales, for example. Another avenue for exploration is to evaluate *what type* of data patterns are specific DR algorithms good at preserving, and then *combining* multiple low-dimensional representations of the same data in order to provide a less biased, more comprehensive view of the projected dataset.

Tying visual patterns to specific data patterns: In Chapter 4, we develop a technique to control the visual patterns showing up in a projection. Our algorithm, ShaRP, can successfully shape the clusters that form in the projection into ellipses, triangles, or rectangles. An interesting line of research would be to choose *a priori* which data patterns we would like to have mapped to which visual patterns. For example, cluster size is *not* in general related to a data pattern — *e.g.*, number of points, within-class variability — in projections. Would it be possible to develop a technique that is guaranteed to map data patterns such as pairwise distances, class variability, or

point density into visual patterns such as projected distances, cluster sizes or point overlap? Initial ideas for tackling this problem could involve, for example, penalizing how much a technique warps the data space in order to produce a projection. In Neural Dimensionality Reduction, this could be achieved by introducing penalties related to the magnitude of the Jacobian of the computed transformation, or to its variation, *i.e.*, the Hessian.

Projection discovery: We explore in this work two options for using Neural Networks to generate projections. In Chapter 3 we propose (S)APPA: these techniques are able to generate projections by learning to *mimic* a reference technique — *e.g.*, t-SNE or UMAP. Its main application is to allow the fast projection of a huge dataset by first projecting a small sample thereof and using a learned approximator to project the rest. In Chapter 4 we describe ShaRP, a DR technique that uses a neural network in order to generate a projection without mimicking. However, ShaRP requires the use of (pseudo)label information, otherwise it generates poor projections showing intermixed clusters and scoring low in terms of quality metrics. When using pseudolabels, ShaRP is highly reliant on the quality of the clustering producing them, a limitation that can easily lead to bad projections. A line of future work is to understand *how* one can build a Neural technique that is able to produce, at the same time, projections that have high quality — *i.e.*, show known data patterns if those exist, score well in terms of quality metrics — *without* the need for a reference projection to mimic or label information. In other words: What kind of inductive bias must a Neural Network have in order to produce *human-readable, informative* projections in fully unsupervised fashion? How do we enforce this inductive bias in terms of neural architecture and loss functions? We believe that placing suitable priors on the latent space — in the vein of APPA’s design — is crucial to enable these powerful statistical models to achieve this goal.

Interpretability of enhanced decision maps: In this thesis’ Chapter 6, we made contributions to the usability and usefulness of Enhanced Decision Maps with our ∂ DBMs. The addition of information that allows a user to ‘locate’ themselves in the data space through this 2D ‘lens’ improves qualitatively the conclusions that can be drawn about a classifier’s behavior. However, the combination of two complex objects — the inverse projection P^{-1} and the classifier —, at least one of which is highly non-linear, still hinders full trust and understanding of exactly what is being displayed. Not only that, but P^{-1} is fully dependent on the quality of the projection it is trained to invert. We believe next steps in improving the ∂ DBM and its family of techniques are evident. Firstly, since projection algorithms each have their own visual signature that stays somewhat constant across datasets — see Section 4.1 — what is the effect of using an algorithm like UMAP, which tightly packs clusters and separates them with a lot of white space,

versus t-SNE, which tends to rather uniformly fill the 2D space with jigsaw-like clusters? This surely has an impact on the computed P^{-1} : In order to properly ‘undo’ UMAP, it would have to stretch the compressed space UMAP creates within clusters, and compress the white space. This would lead to high gradients within clusters, which might affect the accuracy of the created DBM. Another way to improve DBMs would be through interpretable interaction. Recently there have been proposed techniques that allow a user to *move* the DBM in the data space in order to explore the classifier’s behavior in specific areas. Exactly *how* to interpret and control this movement in the data space in a way that is interpretable according to the semantics of the data is still unclear. While in the case of image datasets one can manually inspect the output of P^{-1} and have an idea of whether it makes sense, this is not the same for tabular datasets, where judging what constitutes a plausible tuple is in general not possible through inspection alone, requiring specific tools to be developed.

Datasets as data points: The development of APPA in Chapter 3 has significantly improved the state of the art of what is possible in terms of approximating projections. That is, one possesses already a projection algorithm that one wishes to approximate, leading to a regression problem which we tackle with a Neural Network. However, APPA approximates the *outcome* of applying a given projection algorithm to some dataset. Its inner workings in no way approximate the projection algorithm *itself*: While t-SNE (for example) is an iterative algorithm directly optimizing point positions in the projected space, APPA computes a complex function in its Neural Network that directly takes high-dimensional points to their final positions according to t-SNE. This raises the question: is it possible to approximate the t-SNE *procedure* in a parametric way? In other words, can one *abstract over* the dataset itself and learn some parametric function that can reproduce the effects of t-SNE or any other DR algorithm for a *new, completely unseen* dataset? And what kinds of restrictions would apply to this new dataset so as to enable its meaningful projection through this approximation scheme? This might require interpreting a dataset in terms of the similarity structures present therein — *e.g.*, as a graph of relationships between data points. Another useful abstraction might be the *hypernetwork* — *i.e.*, a neural network that predicts weights for another neural network — where the former ingests the dataset to be projected and adapts the weights of the latter in order to produce a correct projection. In this line of work, a dataset is interpreted as if it was a data *point* instead in order to approximate a DR algorithm, which is itself dataset-agnostic.

APPENDIX

A.1 APPROXIMATING PARAMETRIC PROJECTION ALGORITHM

We present below additional results concerning our methods presented in Chapter 3. We use APPA and SAPPa in additional test cases and include further comparisons with Dimensionality Reduction techniques and their NNP approximations. We also report additional quality metrics which we omitted from Chapter 3 for brevity. These additional results further confirm (S)APPA’s ability to produce approximate projections that strongly reduce the diffusion effect of NNP.

A.1.1 *Visual comparison*

Figure A.1 shows additional results of running (S)APPA for more combinations of dataset and projections. We see further evidence that (S)APPA are better at preserving clusters in their approximate projections than NNP. This is conditional on there being discernible clusters in the training projection — leftmost column — to begin with. For example, for the IMDB dataset and its t-SNE projection (row 1) all of NNP, APPA, and SAPPa fail to create clusters given that there are no clusters in the original projection.

A.1.2 *Dataset descriptions*

We evaluate APPA and SAPPa across ten different datasets. We provide a short description of each of them. Especially noteworthy are datasets on which we perform pre-processing using pre-trained neural network features.

Cats and Dogs: This dataset contains 25K images of two classes: cats and dogs (Elson et al., 2007). We pre-process these images by extracting features from the penultimate layer of a pre-trained Inception V3 Network, following Espadoto et al. (2019a), obtaining 2048-dimensional vectors.

CIFAR10: This dataset contains 60K images of 10 different classes, of which we keep two: Dogs and Automobiles (Krizhevsky, 2009). We extract features using a pre-trained DenseNet, obtaining 1920-dimensional vectors.

CIFAR100: Dataset containing 60K images of 100 different classes, of which we again keep two: Poppies and Bowls (Krizhevsky, 2009). The feature extraction pipeline is the same as for CIFAR10.

FashionMNIST: Dataset of 60K grayscale images of 10 classes of pieces of clothing and accessories (Xiao et al., 2017). Each image is 28 by 28 pixels.

HAR: UCI Human Action Recognition Using Smartphones dataset (Anguita et al., 2012). Contains 7352 training samples, each of which is a 561-dimensional vector.

IMDB: IMDB Movie Review dataset, containing 25K movie reviews with 500 features each (Maas et al., 2011).

MNIST: Dataset of 28 by 28 grayscale images of handwritten digits (10 classes) (LeCun et al., 1998). Contains 60K images.

Reuters: Reuters newswire classification dataset, containing 8432 training samples (Thoma, 2017). We keep 6 out of the original 46 classes and encode them into 5000-dimensional datasets using TF-IDF.

Spambase: Dataset for e-mail spam classification tasks (Hopkins et al., 1999). Contains 4601 training samples where each one is a 57-dimensional vector.

USPS: US Postal Service dataset containing handwritten digits (10 classes). Each image is a 16 by 16 grayscale image (Hull, 1994).

A.1.3 *Additional projection quality metrics*

We evaluate our work according to 15 standard projection quality metrics measuring different aspects of (S)APPA, NNP, and reference projections. Figure A.2 extends the results from Figure 3.5 by the following eight projection quality metrics not shown in Chapter 3: Normalized Stress, Class-Aware Continuity, Class-Aware Trustworthiness, False Neighbor Rate, Pearson Correlation, Shepard Goodness, Jaccard, and Mean Relative Rank Errors Data/Projection.

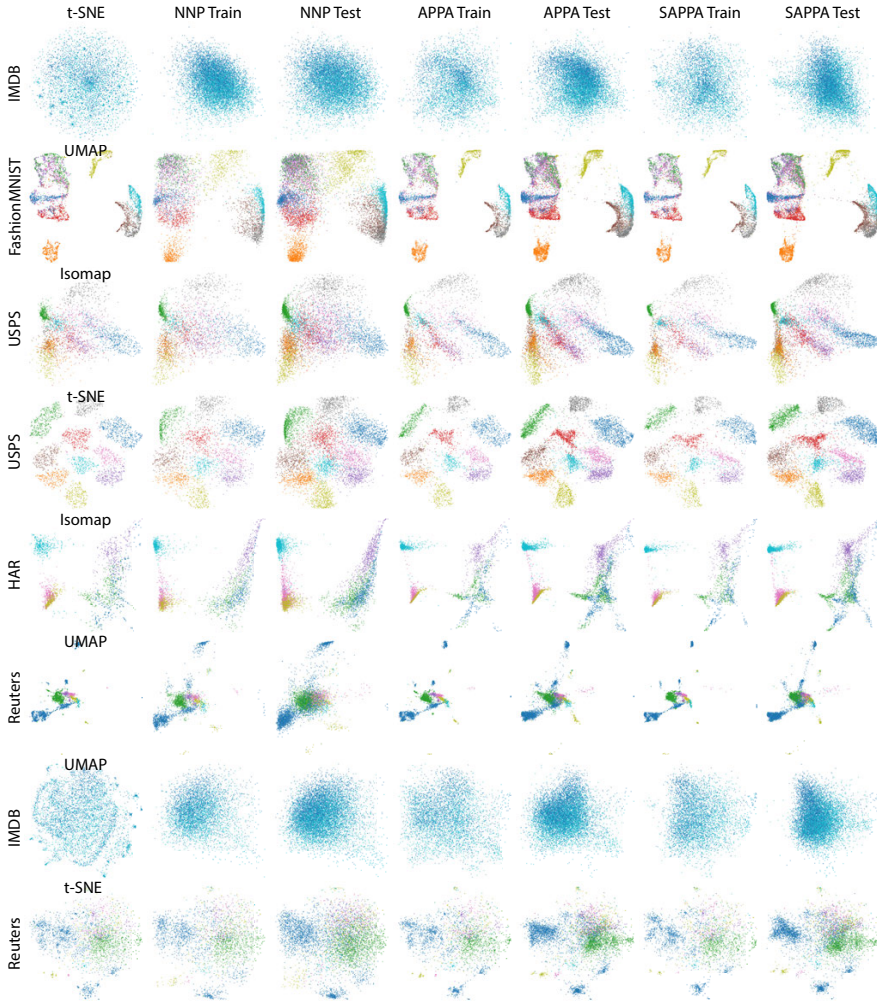


Figure A.1: Additional test cases for comparing APPA, SAPP, and NNP in terms of their approximation quality. There is a clear trend where APPA and SAPP both produce sharper approximate projections than NNP.

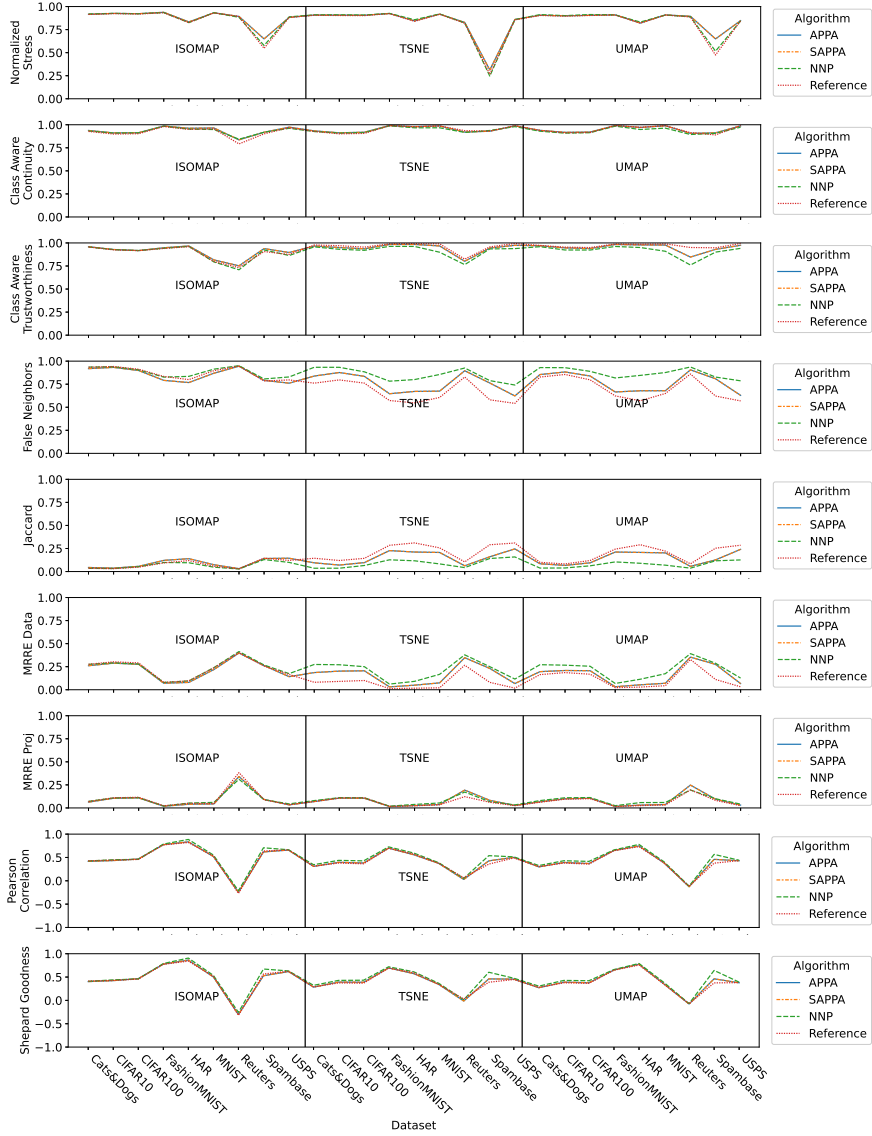


Figure A.2: Additional Projection Quality Metrics. We show the behavior of Normalized Stress, Class-Aware Continuity and Trustworthiness, Pearson Correlation of Pairwise Distances, Shepard Goodness, and False Neighbor Rate. The metrics display the same ‘sandwiching’ behavior as noted in Chapter 3: the Reference projection has best quality, then (S)APPA, then NNP.

A.1.4 *Neural network architecture*

We use the same backbone network for NNP, APPA, and SAPPa, which is slightly different from NNP’s original architecture. Our backbone has more capacity than the original NNP one which had three layers with sizes 256, 512, and 256. We use a single architecture to ensure comparability — varying the architecture would introduce another possible cause for differences between (S)APPA and NNP. Our architecture is as follows:

```
import torch.nn as nn

net = nn.Sequential(
    nn.Dropout(p=0.5),
    nn.Linear(input_dim, 2048),
    nn.SiLU(),
    nn.Linear(2048, 2048),
    nn.SiLU(),
    nn.Linear(2048, 2048),
    nn.SiLU(),
    nn.Linear(2048, 2),
    nn.Sigmoid(),
)
```

As the listing above shows, we use 3 layers in our neural network plus one output layer (where we assume we project to 2 dimensions). For the hidden layers, we use the Sigmoid Linear Unit (SiLU) activation function ([Hendrycks and Gimpel, 2016](#)). The reference projection $\mathbf{Y}$ is assumed to be scaled to $[0, 1]^2$, so we use a Sigmoid activation function $\sigma(x) = \frac{1}{1+\exp(-x)}$ in the output layer.

A.2 SHAPE-REGULARIZED PROJECTION

We present below additional results for ShaRP, the Dimensionality Reduction method proposed in Chapter 4. These consist of using the main shape regularization targets — Ellipses, Rectangles, and Triangles — for an extended collection of datasets, as well as quality metric values from all our experiments.

A.2.1 *Quality Metrics*

The raw values of all metrics used to evaluate projections in our work are listed in Table 13. Particularly striking are the high values for Stress (Q_s) for SSNP in datasets with only two classes. For a discussion on why this happens, see the main text.

Table 13: Quality metrics for all studied combinations of projection algorithm and dataset. Spans multiple pages.

Projection	Dataset	Trustworthiness	Continuity	Shepard Goodness	Stress	Neighb. Hit	Dist. Cons.	Train MSE ¹	Test MSE ¹
Isomap	Bank	0.795	0.925	0.747	0.983	0.883	0.590	—	—
	CNAE-9	0.662	0.873	0.615	0.911	0.660	0.437	—	—
	COIL20	0.867	0.988	0.725	2.890	0.775	0.534	—	—
	FMD	0.748	0.901	0.738	0.980	0.536	0.309	—	—
	F-MNIST	0.920	0.976	0.874	1.934	0.685	0.544	—	—
	HAR	0.924	0.971	0.950	3.467	0.863	0.729	—	—
	HateSpeech	0.700	0.930	0.480	1.070	0.822	0.430	—	—
	IMDB	0.548	0.578	0.524	0.536	0.678	0.538	—	—
	MNIST	0.760	0.959	0.763	1.571	0.621	0.490	—	—
	QuickDraw	0.712	0.917	0.732	1.233	0.468	0.247	—	—
	Reuters	0.625	0.756	0.423	1.354	0.753	0.540	—	—
	SVHN	0.857	0.941	0.928	0.789	0.427	0.099	—	—
	Sentiment	0.649	0.893	0.552	0.876	0.706	0.497	—	—
	SpamBase	0.752	0.947	0.761	0.572	0.876	0.807	—	—
t-SNE	USPS	0.855	0.973	0.840	1.310	0.762	0.633	—	—
	Bank	0.978	0.923	0.761	7.608	0.890	0.658	—	—
	CNAE-9	0.820	0.933	0.600	7.661	0.879	0.509	—	—
	COIL20	0.996	0.994	0.777	4.291	0.990	0.733	—	—
	FMD	0.874	0.894	0.716	1.447	0.776	0.379	—	—
	F-MNIST	0.990	0.987	0.830	2.361	0.842	0.654	—	—
	HAR	0.991	0.986	0.782	6.669	0.967	0.820	—	—
	HateSpeech	0.977	0.940	0.550	25.439	0.854	0.167	—	—
	IMDB	0.586	0.768	0.610	0.454	0.695	0.591	—	—
	MNIST	0.986	0.972	0.701	3.024	0.945	0.768	—	—
	QuickDraw	0.910	0.896	0.733	1.275	0.561	0.261	—	—
	Reuters	0.742	0.912	0.510	5.551	0.846	0.580	—	—
	SVHN	0.916	0.946	0.888	0.931	0.454	0.114	—	—
	Sentiment	0.940	0.892	0.614	23.639	0.774	0.569	—	—
	SpamBase	0.966	0.964	0.710	63.257	0.926	0.767	—	—
	USPS	0.989	0.984	0.749	5.996	0.969	0.940	—	—

Continues on next page

¹ Only for invertible projections.

A.2 SHAPE-REGULARIZED PROJECTION

Projection	Dataset	Trustworthiness	Continuity	Shepard Goodness	Stress	Neighb. Hit	Dist. Cons.	Train MSE ¹	Test MSE ¹
UMAP	Bank	0.946	0.939	0.710	0.647	0.888	0.545	—	—
	CNAE-9	0.794	0.943	0.580	1.380	0.845	0.604	—	—
	COIL20	0.994	0.996	0.650	1.256	0.924	0.805	—	—
	FMD	0.837	0.913	0.596	0.620	0.721	0.436	—	—
	F-MNIST	0.981	0.988	0.823	0.415	0.803	0.668	—	—
	HAR	0.979	0.989	0.850	1.910	0.931	0.743	—	—
	HateSpeech	0.972	0.954	0.436	9.227	0.845	0.307	—	—
	IMDB	0.517	0.599	0.521	0.469	0.668	0.518	—	—
	MNIST	0.959	0.974	0.690	0.484	0.921	0.835	—	—
	QuickDraw	0.823	0.926	0.712	0.728	0.500	0.276	—	—
	Reuters	0.659	0.890	0.452	1.306	0.756	0.525	—	—
	SVHN	0.898	0.951	0.890	0.535	0.438	0.099	—	—
	Sentiment	0.926	0.922	0.390	9.190	0.758	0.512	—	—
	SpamBase	0.937	0.965	0.680	13.118	0.900	0.767	—	—
	USPS	0.972	0.985	0.708	1.068	0.960	0.952	—	—
NNP[t-SNE]	Bank	0.924	0.922	0.754	0.879	0.881	0.638	—	—
	CNAE-9	0.775	0.929	0.600	0.860	0.800	0.645	—	—
	COIL20	0.989	0.993	0.773	0.945	0.945	0.735	—	—
	FMD	0.759	0.874	0.702	0.949	0.573	0.336	—	—
	F-MNIST	0.967	0.986	0.837	0.961	0.769	0.653	—	—
	HAR	0.967	0.985	0.806	0.919	0.914	0.811	—	—
	HateSpeech	0.863	0.940	0.552	0.725	0.829	0.187	—	—
	IMDB	0.570	0.757	0.605	0.893	0.690	0.551	—	—
	MNIST	0.942	0.971	0.717	0.955	0.881	0.742	—	—
	QuickDraw	0.779	0.894	0.742	0.966	0.480	0.258	—	—
	Reuters	0.687	0.909	0.513	0.906	0.796	0.586	—	—
	SVHN	0.871	0.936	0.886	0.951	0.406	0.141	—	—
	Sentiment	0.810	0.895	0.616	0.725	0.724	0.541	—	—
	SpamBase	0.874	0.965	0.723	0.493	0.888	0.786	—	—
	USPS	0.951	0.982	0.738	0.925	0.947	0.933	—	—
AE	Bank	0.948	0.876	0.617	3.945	0.893	0.602	0.051	0.055
	CNAE-9	0.635	0.800	0.603	1.262	0.689	0.549	0.004	0.004
	COIL20	0.976	0.983	0.749	3.612	0.906	0.643	0.019	0.022
	FMD	0.804	0.868	0.677	0.855	0.559	0.343	0.014	0.014
	F-MNIST	0.967	0.977	0.761	0.534	0.746	0.627	0.028	0.029
	HAR	0.944	0.973	0.911	0.507	0.819	0.657	0.006	0.006
	HateSpeech	0.816	0.843	0.395	6.019	0.832	0.677	0.006	0.006
	IMDB	0.560	0.536	0.517	0.592	0.772	0.740	0.006	0.006
	MNIST	0.921	0.917	0.493	1.637	0.752	0.498	0.039	0.042
	QuickDraw	0.816	0.744	0.515	0.725	0.459	0.203	0.066	0.068
	Reuters	0.642	0.768	0.579	1.919	0.814	0.699	0.001	0.001
	SVHN	0.873	0.925	0.890	0.814	0.408	0.126	0.015	0.016
	Sentiment	0.666	0.790	0.338	3.336	0.714	0.565	0.004	0.004
	SpamBase	0.834	0.916	0.629	7.259	0.919	0.791	0.001	0.002
	USPS	0.928	0.957	0.607	0.918	0.840	0.656	0.028	0.029
SSNP(AG)	Bank	0.677	0.799	0.683	2.8e+05	0.882	0.712	0.186	0.183
	CNAE-9	0.661	0.850	0.673	1.051	0.655	0.498	0.004	0.004
	COIL20	0.958	0.982	0.773	1.926	0.852	0.640	0.032	0.035
	FMD	0.743	0.840	0.711	0.586	0.517	0.252	0.015	0.015
	F-MNIST	0.937	0.972	0.840	0.377	0.759	0.616	0.038	0.039
	HAR	0.924	0.965	0.898	0.990	0.839	0.689	0.008	0.008
	HateSpeech	0.703	0.852	0.589	0.832	0.818	0.534	0.008	0.008
	IMDB	0.528	0.389	0.461	8.3e+06	0.666	0.521	0.028	0.028
	MNIST	0.837	0.922	0.553	0.450	0.804	0.666	0.051	0.052
	QuickDraw	0.740	0.823	0.714	0.817	0.470	0.253	0.073	0.074

Continues on next page

APPENDIX

Projection	Dataset	Trustworthiness	Continuity	Shepard Goodness	Stress	Neighb. Hit	Dist. Cons.	Train MSE ¹	Test MSE ¹
	Reuters	0.610	0.774	0.610	0.481	0.752	0.588	0.001	0.001
	SVHN	0.847	0.841	0.744	0.688	0.409	0.126	0.017	0.020
	Sentiment	0.610	0.640	0.339	2.4e+06	0.679	0.541	0.014	0.014
	SpamBase	0.610	0.817	0.675	4.1e+07	0.746	0.663	0.070	0.070
	USPS	0.882	0.952	0.673	0.810	0.851	0.726	0.037	0.038
SSNP(KM)	Bank	0.675	0.801	0.682	2.8e+05	0.883	0.712	0.211	0.210
	CNAE-9	0.657	0.823	0.650	0.946	0.606	0.384	0.004	0.004
	COIL20	0.948	0.975	0.729	1.961	0.831	0.585	0.035	0.036
	FMD	0.761	0.863	0.695	0.474	0.567	0.293	0.015	0.015
	F-MNIST	0.951	0.979	0.825	0.699	0.740	0.573	0.034	0.035
	HAR	0.927	0.970	0.946	0.964	0.804	0.703	0.008	0.008
	HateSpeech	0.717	0.796	0.561	0.720	0.812	0.404	0.008	0.008
	IMDB	0.528	0.389	0.461	8.2e+06	0.672	0.522	0.026	0.026
	MNIST	0.874	0.910	0.539	0.501	0.763	0.349	0.050	0.052
	QuickDraw	0.768	0.821	0.728	0.745	0.464	0.229	0.070	0.072
	Reuters	0.617	0.828	0.526	0.600	0.761	0.649	0.001	0.001
	SVHN	0.848	0.896	0.845	0.594	0.421	0.123	0.019	0.021
	Sentiment	0.612	0.641	0.341	2.3e+06	0.691	0.531	0.024	0.024
	SpamBase	0.611	0.817	0.675	4.1e+07	0.743	0.663	0.004	0.004
	USPS	0.906	0.962	0.727	0.648	0.850	0.776	0.036	0.037
SSNP(GT)	Bank	0.684	0.804	0.688	2.9e+05	0.872	0.712	0.213	0.212
	CNAE-9	0.656	0.864	0.658	1.801	0.995	0.962	0.004	0.004
	COIL20	0.924	0.982	0.789	2.609	0.970	0.724	0.035	0.036
	FMD	0.661	0.832	0.601	0.545	0.776	0.685	0.020	0.020
	F-MNIST	0.869	0.950	0.737	0.417	0.885	0.727	0.047	0.047
	HAR	0.878	0.953	0.906	0.502	0.981	0.961	0.009	0.009
	HateSpeech	0.669	0.766	0.502	0.788	0.927	0.886	0.009	0.009
	IMDB	0.527	0.389	0.462	8.9e+06	0.670	0.530	0.026	0.026
	MNIST	0.782	0.929	0.677	0.435	0.987	0.886	0.053	0.054
	QuickDraw	0.613	0.727	0.577	0.876	0.945	0.788	0.080	0.080
	Reuters	0.576	0.767	0.717	0.503	0.977	0.972	0.001	0.002
	SVHN	0.634	0.715	0.625	0.911	0.474	0.246	0.040	0.044
	Sentiment	0.617	0.664	0.356	2.1e+06	0.689	0.530	0.005	0.005
	SpamBase	0.612	0.816	0.673	4.3e+07	0.745	0.655	0.004	0.004
	USPS	0.856	0.961	0.749	0.397	0.987	0.920	0.039	0.040
ShaRP(AG)	Bank	0.598	0.643	0.586	0.815	0.874	0.502	0.090	0.088
	CNAE-9	0.638	0.817	0.576	0.768	0.614	0.465	0.004	0.004
	COIL20	0.916	0.955	0.652	0.895	0.755	0.596	0.037	0.036
	FMD	0.622	0.638	0.718	0.920	0.415	0.149	0.017	0.016
	F-MNIST	0.923	0.956	0.857	0.933	0.719	0.616	0.039	0.040
	HAR	0.879	0.911	0.779	0.857	0.797	0.726	0.008	0.008
	HateSpeech	0.545	0.603	0.550	0.606	0.763	0.382	0.009	0.009
	IMDB	0.509	0.591	0.516	0.783	0.672	0.580	0.006	0.006
	MNIST	0.820	0.912	0.594	0.927	0.786	0.666	0.053	0.053
	QuickDraw	0.673	0.777	0.642	0.936	0.461	0.203	0.076	0.076
	Reuters	0.583	0.696	0.565	0.816	0.734	0.592	0.001	0.001
	SVHN	0.667	0.572	0.728	0.933	0.412	0.103	0.027	0.028
	Sentiment	0.511	0.517	0.503	0.577	0.676	0.505	0.004	0.005
	SpamBase	0.503	0.512	0.490	0.772	0.689	0.508	0.003	0.003
	USPS	0.863	0.938	0.705	0.876	0.834	0.709	0.039	0.040

Continues on next page

A.2 SHAPE-REGULARIZED PROJECTION

Projection	Dataset	Trustwor- thiness	Continuity	Shepard Goodness	Stress	Neighb. Hit	Dist. Cons.	Train MSE ¹	Test MSE ¹
ShaRP(KM)	Bank	0.613	0.662	0.604	0.823	0.863	0.516	0.088	0.088
	CNAE-9	0.611	0.709	0.493	0.752	0.570	0.269	0.004	0.004
	COIL20	0.931	0.958	0.712	0.894	0.747	0.596	0.036	0.036
	FMD	0.630	0.619	0.738	0.935	0.398	0.123	0.017	0.017
	F-MNIST	0.936	0.960	0.862	0.933	0.704	0.598	0.037	0.038
	HAR	0.893	0.918	0.774	0.864	0.757	0.668	0.009	0.009
	HateSpeech	0.561	0.559	0.527	0.561	0.764	0.383	0.009	0.009
	IMDB	0.522	0.576	0.507	0.784	0.664	0.507	0.006	0.006
	MNIST	0.856	0.909	0.660	0.925	0.731	0.562	0.051	0.051
	QuickDraw	0.723	0.786	0.614	0.938	0.448	0.222	0.073	0.074
	Reuters	0.585	0.766	0.578	0.816	0.728	0.635	0.001	0.001
	SVHN	0.673	0.571	0.723	0.921	0.400	0.108	0.026	0.027
	Sentiment	0.526	0.518	0.502	0.602	0.683	0.490	0.004	0.005
	SpamBase	0.504	0.512	0.501	0.760	0.688	0.498	0.003	0.003
	USPS	0.886	0.938	0.692	0.876	0.817	0.660	0.039	0.041
ShaRP(GT)	Bank	0.511	0.497	0.488	0.837	0.971	0.517	0.097	0.096
	CNAE-9	0.624	0.820	0.557	0.745	0.977	0.975	0.004	0.004
	COIL20	0.855	0.955	0.634	0.893	0.963	0.956	0.043	0.042
	FMD	0.622	0.793	0.547	0.912	0.893	0.876	0.018	0.018
	F-MNIST	0.803	0.901	0.698	0.935	0.882	0.856	0.052	0.052
	HAR	0.814	0.884	0.722	0.871	0.954	0.945	0.010	0.010
	HateSpeech	0.518	0.558	0.529	0.590	0.884	0.543	0.009	0.009
	IMDB	0.509	0.537	0.509	0.781	0.994	0.993	0.006	0.006
	MNIST	0.727	0.887	0.566	0.933	0.970	0.938	0.056	0.056
	QuickDraw	0.578	0.673	0.568	0.936	0.932	0.874	0.082	0.081
	Reuters	0.551	0.648	0.550	0.819	0.956	0.760	0.001	0.001
	SVHN	0.511	0.534	0.524	0.939	0.402	0.109	0.044	0.049
	Sentiment	0.508	0.609	0.558	0.580	0.893	0.882	0.004	0.005
	SpamBase	0.542	0.648	0.545	0.763	0.979	0.981	0.003	0.004
	USPS	0.792	0.913	0.630	0.885	0.968	0.939	0.045	0.046

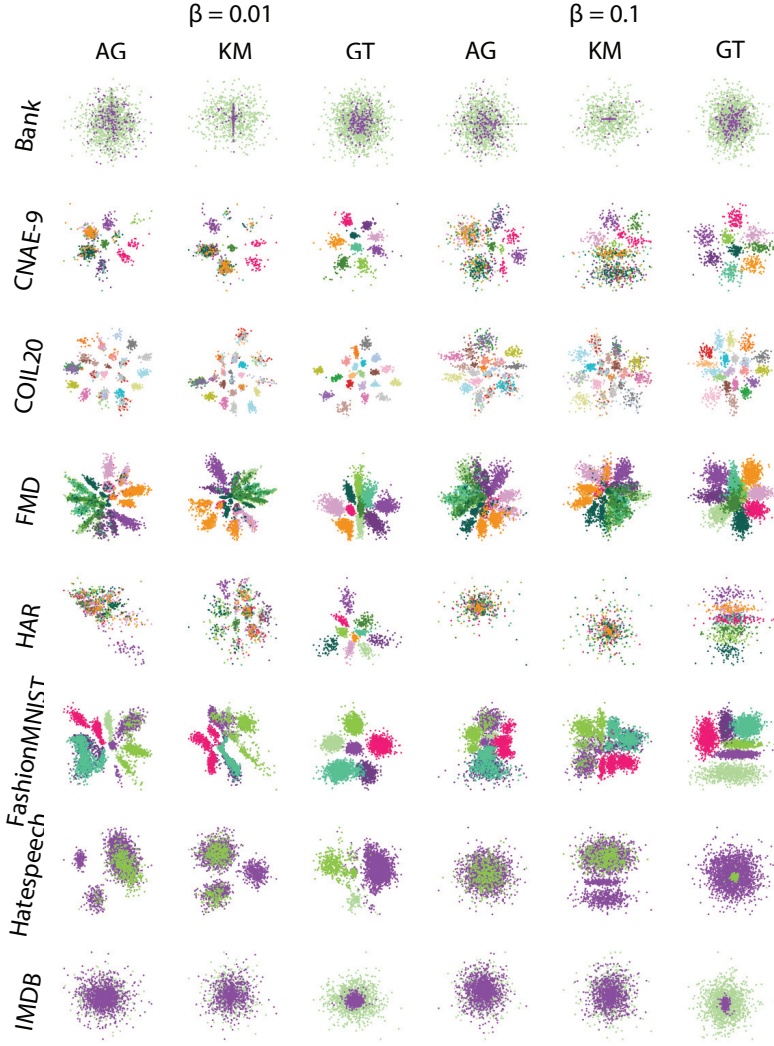
A.2.2 *Shaping 2D projections: ellipses*

Figure A.3: 2D ShaRP using ellipses as shape target for different shape regularization strengths ($\beta \in \{0.01, 0.1\}$). The three variants of ShaRP are shown: AG uses Agglomerative Clustering to derive pseudolabels, KM uses K-Means for the same goal. GT uses the ground truth labels of the dataset.

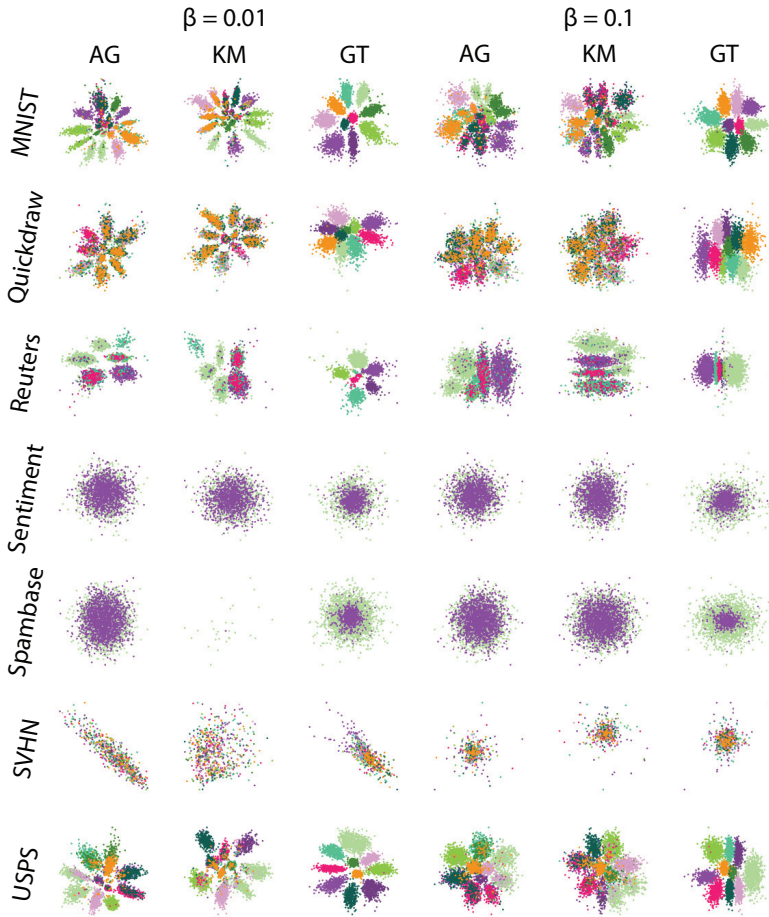


Figure A.4: Continuation of Figure A.3.

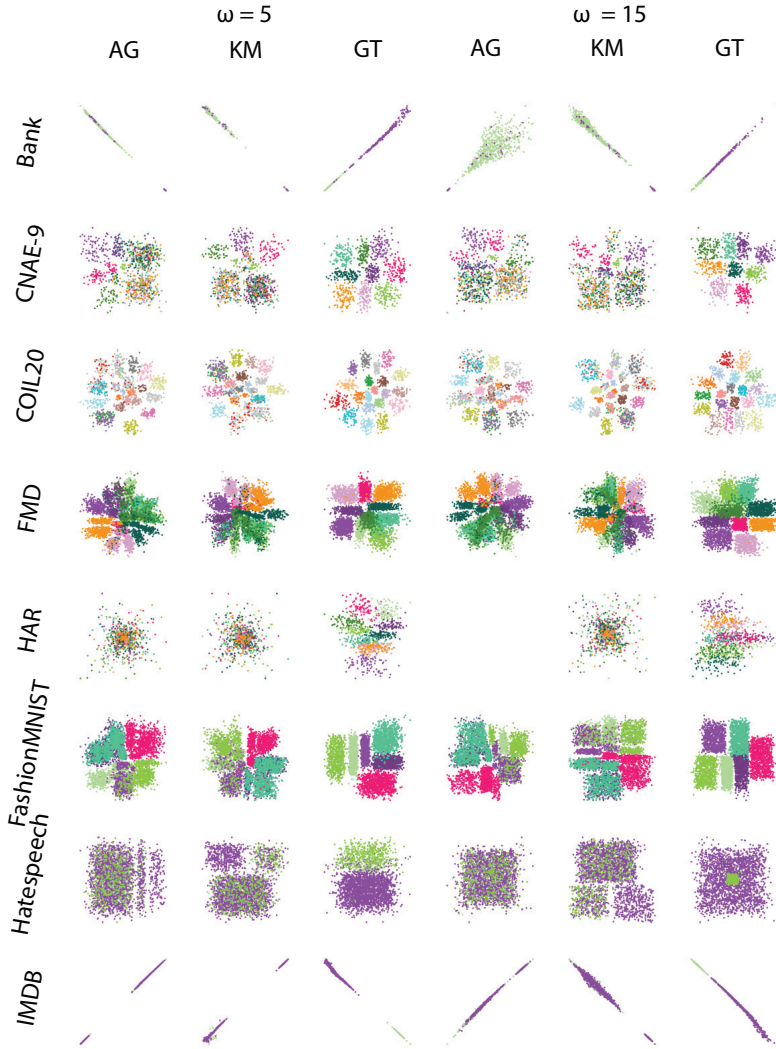
A.2.3 *Shaping 2D projections: rectangles*

Figure A.5: 2D ShaRP using Rectangles as shape target for different values of ω . The three variants of ShaRP are shown: AG uses Agglomerative Clustering to derive pseudolabels, KM uses K-Means for the same goal. GT uses the ground truth labels of the dataset.

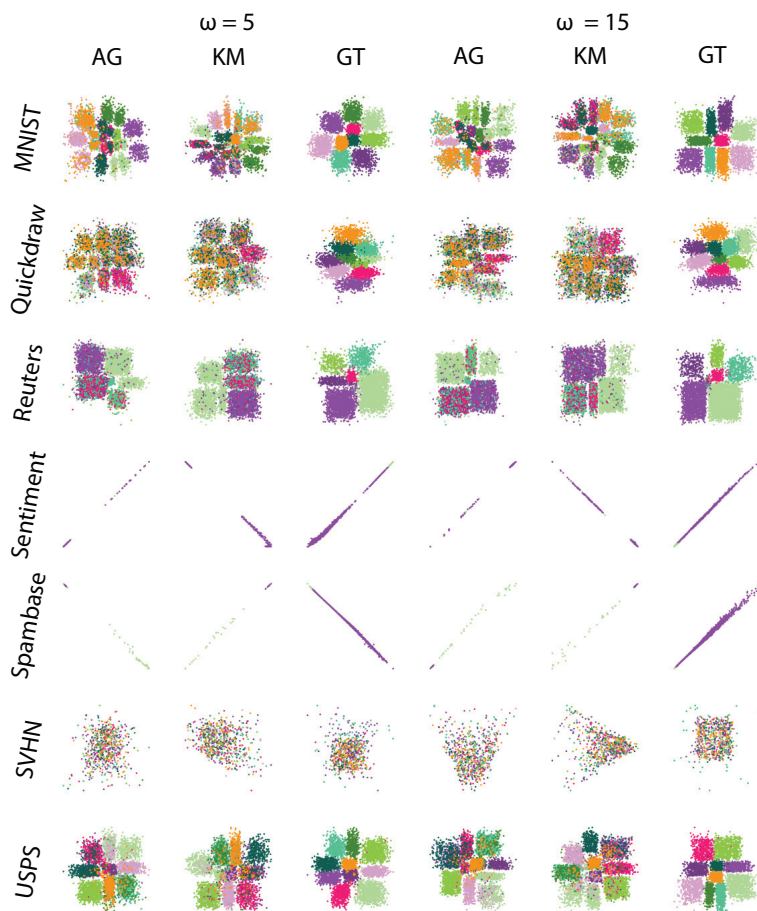


Figure A.6: Continuation of Figure A.5.

A.2.4 Shaping 2D projections: triangles

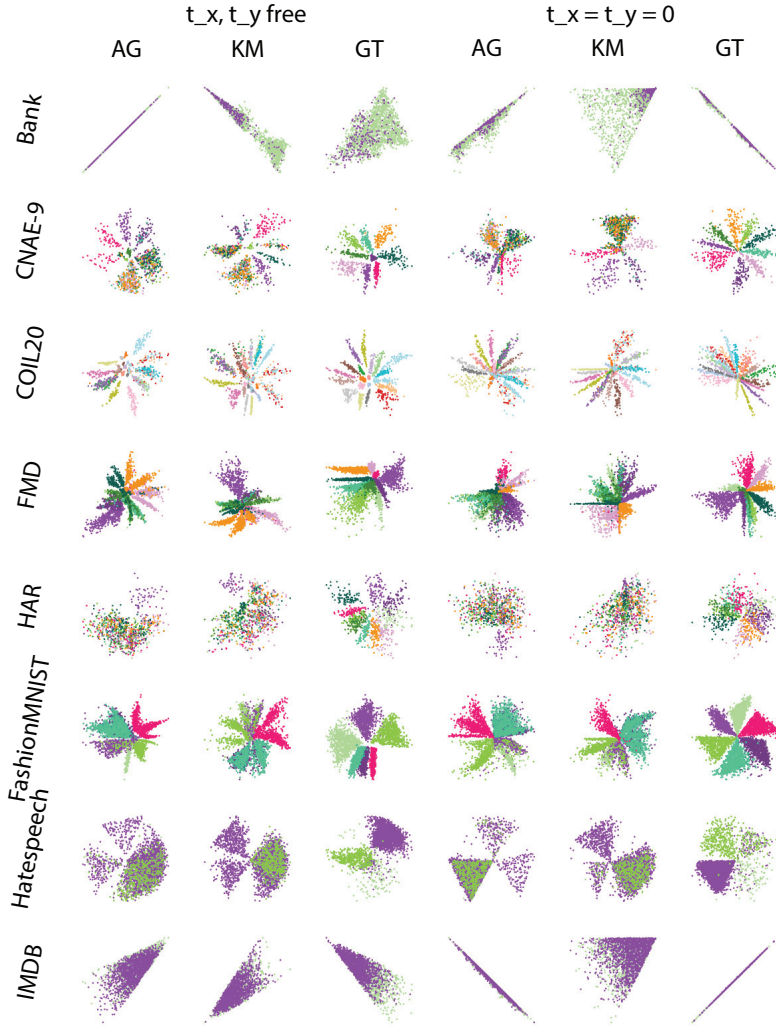


Figure A.7: 2D ShaRP using Triangles as shape target, allowing for translation (left) or not (right). The three variants of ShaRP are shown: AG uses Agglomerative Clustering to derive pseudolabels, KM uses K-Means for the same goal. GT uses the ground truth labels of the dataset.

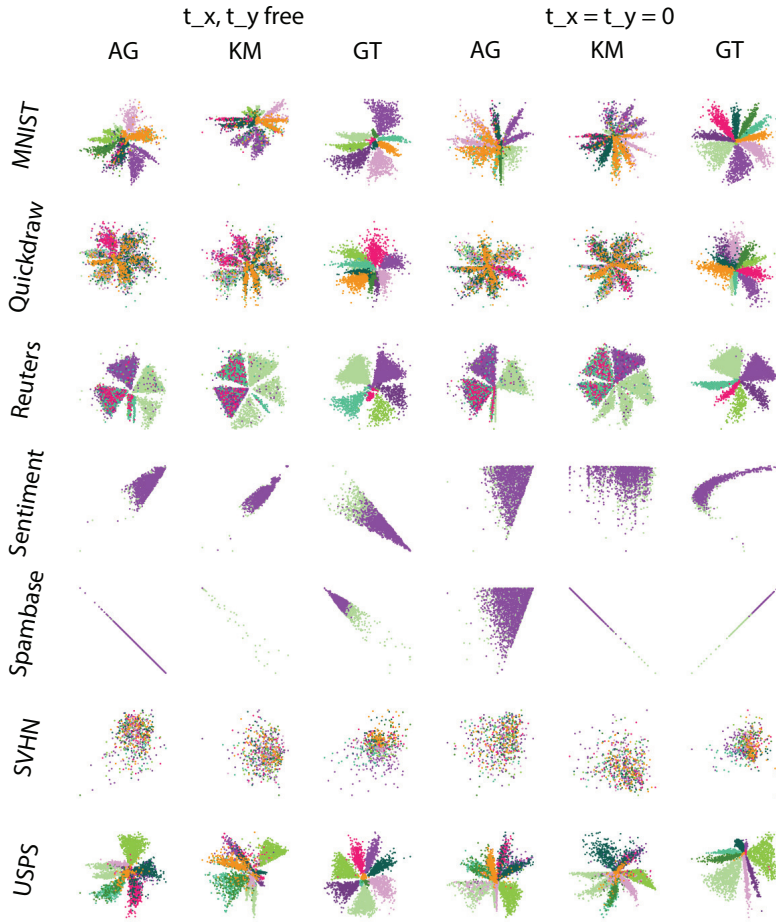


Figure A.8: Continuation of Figure A.7.

A.3 FOOLING PROJECTION QUALITY METRICS

We next present additional results concerning our Metric Fooling pipeline described in Chapter 5.

A.3.1 *Results of pipeline execution*

We demonstrate our pipeline’s ability to handle a wide variety of datasets, projections, and target metrics by including more samples of its produced outputs below in Figures A.9 and A.10. For a comprehensive view of all the results generated by our experiments, we direct the reader to our website at <https://amreis.github.io/fool-proj-metrics>.

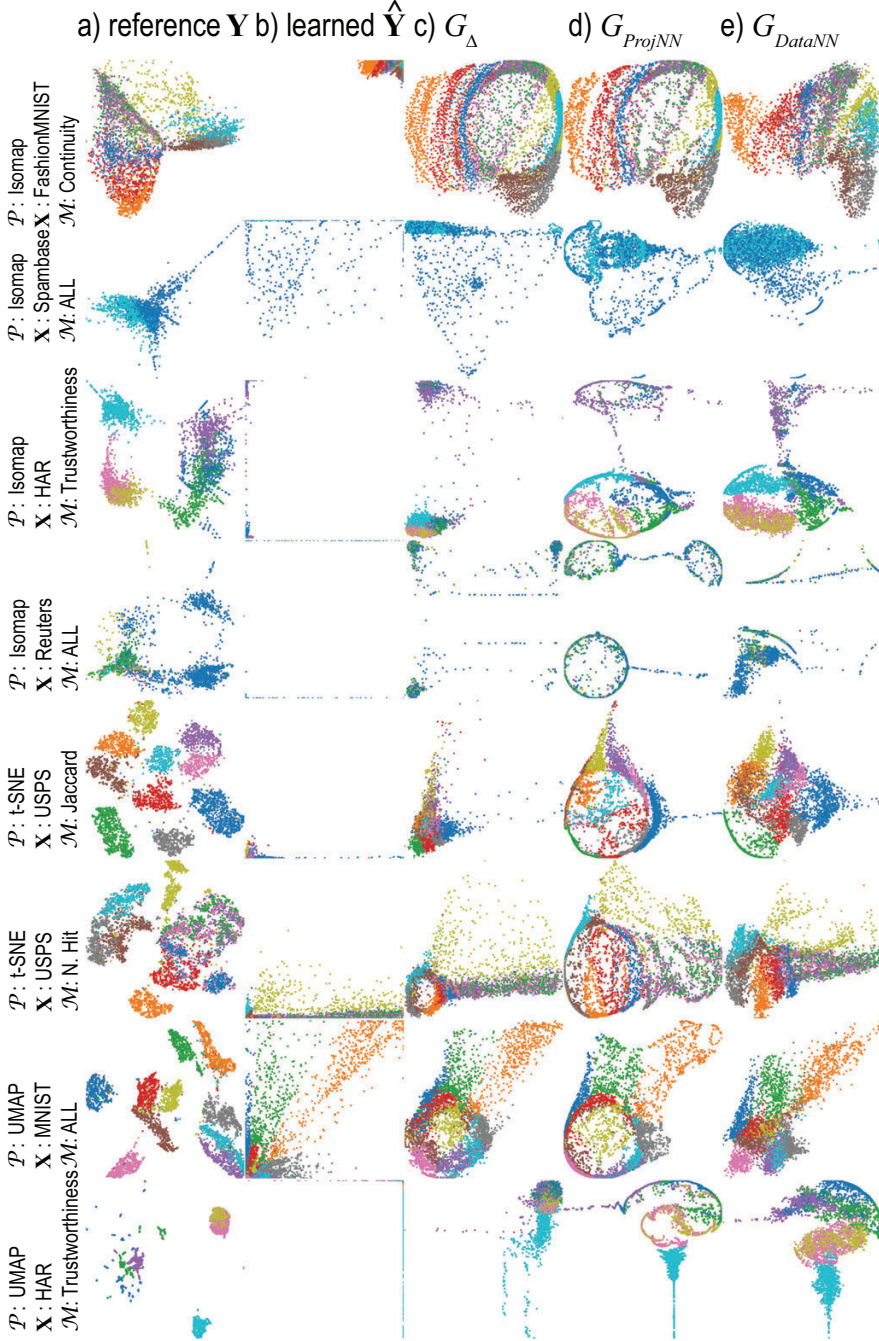


Figure A.9: Reference projections (a), learned 'fooling' projections (b), and the output of our three post-processing algorithms applied to the learned projection (c-e).

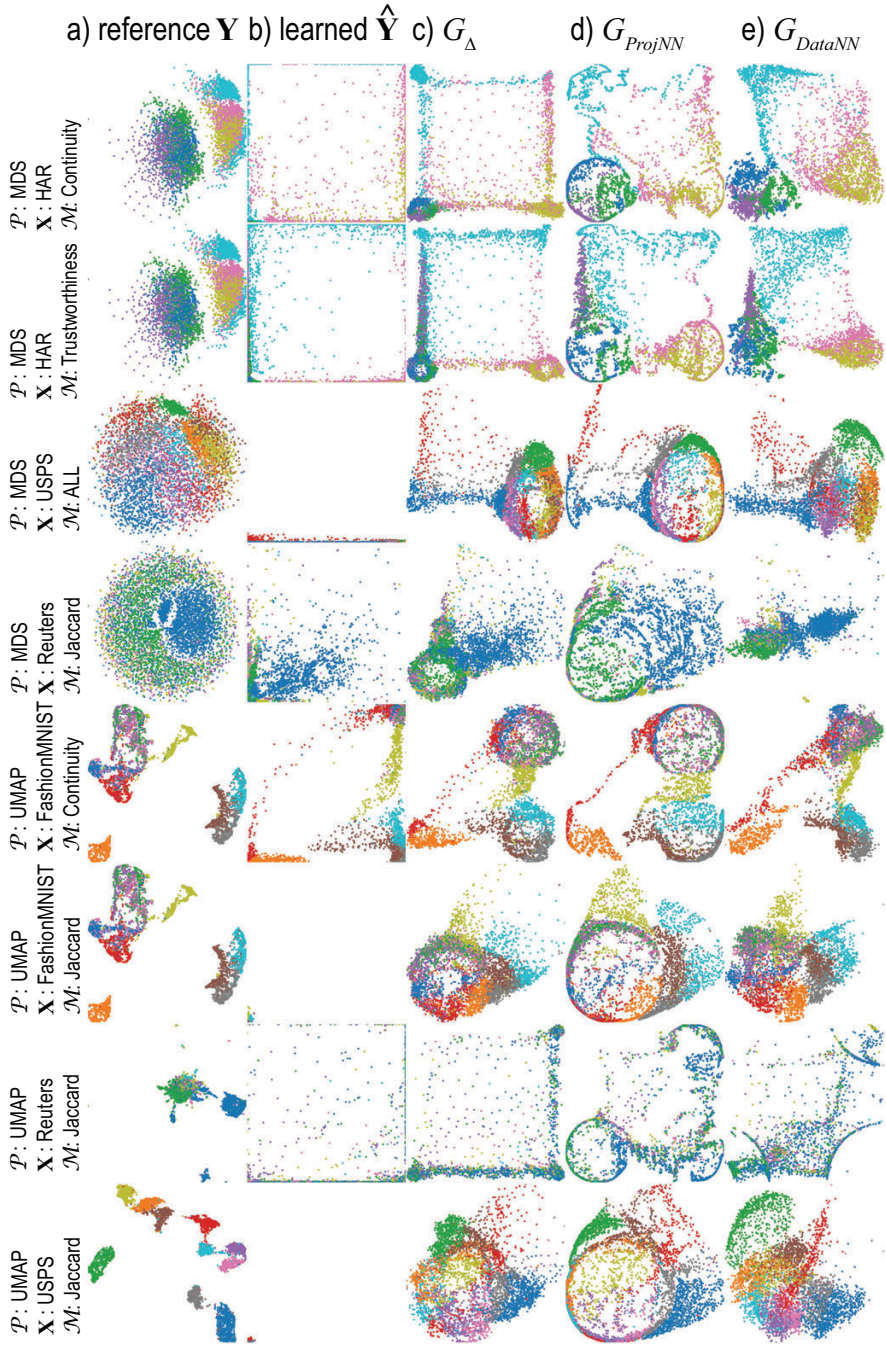


Figure A.10: Continuation of Figure A.9.

A.3.2 Architecture of pipeline steps

Our work uses three neural networks collaborating to generate a ‘fooling’ projection $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$. Here we describe their architectures in more detail. We represent the sigmoid activation function by $\sigma(x) = (1 + e^{-x})^{-1}$, and the Rectified Linear Unit activation function by $\text{ReLU}(x) = \max(0, x)$.

Quality metric approximator: Q_θ receives as input a batch of points from the dataset $\mathbf{X}$ as well as from a reference projection $\mathbf{Y}$ and approximates a local quality metric $\mu = \{\mu_i\}$.

$$\begin{aligned} \mathbf{h}_0 &\doteq [\mathbf{X}, \mathbf{Y}] \\ \mathbf{h}_i &= \text{ReLU}(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i) \quad (i \in \{1, \dots, 4\}) \\ \hat{\mu} &= \sigma(\mathbf{h}_4) \\ \mathcal{L}_Q &= \|\hat{\mu} - \mu\|^2 \end{aligned}$$

with hidden layer sizes of 512, 128, and 32 neurons. $[\mathbf{X}, \mathbf{Y}]$ means to concatenate $\mathbf{X}$ and $\mathbf{Y}$ along the last dimension. All weight matrices are initialized with Kaiming normal initialization, and biases are initialized to 0.

Projection head: P_ϕ aims to transform data points $\mathbf{X}$ into a projection $\mathbf{Y}$. Its training is aimed at maximizing the projection quality as predicted by Q_θ . We also employ two auxiliary goals to stabilize training: for the first 10 epochs, P_ϕ aims to reproduce the given reference projection $\mathbf{Y}$; and throughout training, an additional head $\mathcal{I}_\psi$ tries to reconstruct $\mathbf{X}$ from the learned projection $\hat{\mathbf{Y}} = P_\phi(\mathbf{X})$.

$$\begin{aligned} \mathbf{h}_0 &\doteq \mathbf{X} \\ \mathbf{h}_i &= \text{ReLU}(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i) \quad (i \in \{1, \dots, 4\}) \\ \hat{\mathbf{Y}} &= \sigma(\mathbf{h}_4) \\ \mathcal{L}_P &= -\frac{1}{n} \sum_{i=1}^n \left(Q_\theta(\mathbf{x}_i, P_\phi(\mathbf{x}_i)) - \kappa \|P_\phi(\mathbf{x}_i) - \mathbf{y}_i\|^2 \right. \\ &\quad \left. - \ln(1 + \|\mathcal{I}_\psi(P_\phi(\mathbf{x}_i)) - \mathbf{x}_i\|^2) \right) \end{aligned}$$

with hidden layer sizes of 512, 128, and 32 neurons. We use a sigmoid activation function for this network and normalize both $\mathbf{X}$ and $\mathbf{Y}$ to lie in $[0, 1]^D$ and $[0, 1]^d$, respectively. All weight matrices are initialized with Kaiming uniform initialization, and biases are initialized to 10^{-4} .

Reconstruction head: $\mathcal{I}_\psi$ aims to reconstruct the input data $\mathbf{X}$ (min-max normalized) as a means of stabilizing the learning objective of P_ϕ . Its architecture and loss function are described below.

$$\begin{aligned}\mathbf{h}_0 &\doteq \hat{\mathbf{Y}} \\ \mathbf{h}_i &= \text{ReLU}(\mathbf{W}_i \mathbf{h}_{i-1} + \mathbf{b}_i) \quad (i \in \{1, \dots, 4\}) \\ \hat{\mathbf{X}} &= \sigma(\mathbf{h}_4) \\ \mathcal{L}_{\mathcal{I}} &= \|\hat{\mathbf{X}} - \mathbf{X}\|^2\end{aligned}$$

with hidden layer sizes of 32, 128, and 512 neurons. All weight matrices are initialized with Kaiming uniform initialization, and biases are initialized to 10^{-4} .

A.3.3 Fooling procedure description

We provide a full description of the algorithms used in our fooling procedure. Algorithm 2 is the *destructive* phase of our pipeline. It is aimed at introducing low-quality visual patterns without lowering a target projection quality metric μ . This algorithm needs to train a PQM approximator Q_θ , which is then used to generate the fooling projection $\hat{\mathbf{Y}}$.

Algorithm 3 is designed to introduce fake but plausible patterns into the obtained projection plot, again without influencing the values of μ (as much as possible).

Algorithm 2 Fooling procedure — Destructive phase

Input: data points $\mathbf{X} = \{\mathbf{x}_i\} \in \mathbb{R}^D$, projected points $\mathbf{Y} = \mathcal{P}(\mathbf{X})$, (optional) labels $\mathbf{C} = \{c_i\}$, target local PQM μ

Output: a μ -fooling projection $\hat{\mathbf{Y}}$

 Compute per-data point quality values $\mu_i(\mathbf{X}, \mathbf{Y}, \mathbf{C})$

 Initialize Q_θ

$\theta \leftarrow \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n (Q_\theta(\mathbf{x}_i, \mathbf{y}_i) - \mu_i)^2$ ▷ run for 100 epochs

 Initialize $P_\phi, \mathcal{I}_\psi$

$\kappa \leftarrow 10$

$\phi, \psi \leftarrow \arg \min_{\phi, \psi} -\frac{1}{n} \sum_{i=1}^n \left(Q_\theta(\mathbf{x}_i, P_\phi(\mathbf{x}_i)) \right.$
 $\left. - \kappa \|P_\phi(\mathbf{x}_i) - \mathbf{y}_i\|^2 \right.$ ▷ run for 10 epochs
 $\left. - \ln(1 + \|\mathcal{I}_\psi(P_\phi(\mathbf{x}_i)) - \mathbf{x}_i\|^2) \right)$

 ▷ set $\kappa \leftarrow 0$, run for 990 more epochs (below)

$\phi, \psi \leftarrow \arg \min_{\phi, \psi} -\frac{1}{n} \sum_{i=1}^n \left(Q_\theta(\mathbf{x}_i, P_\phi(\mathbf{x}_i)) \right.$
 $\left. - \ln(1 + \|\mathcal{I}_\psi(P_\phi(\mathbf{x}_i)) - \mathbf{x}_i\|^2) \right)$

$\hat{\mathbf{Y}} \leftarrow P_\phi(\mathbf{X})$

 Discard $Q_\theta, P_\phi, \mathcal{I}_\psi$

return $\hat{\mathbf{Y}}$

Algorithm 3 Fooling procedure — Constructive phase

Input: data points $\mathbf{X} = \{\mathbf{x}_i\} \in \mathbb{R}^D$, fooling projection $\hat{\mathbf{Y}}$, strategy $\Sigma \in \{\text{ProjNN}, \text{DataNN}, \Delta\}$

Output: a post-processed projection $\hat{\mathbf{Y}}_{\text{Post}}$ containing more plausible visual patterns

if $\Sigma = \text{ProjNN}$ **then**

$G \leftarrow \text{UnweightedKNNGraph}(\hat{\mathbf{Y}}, k = 7)$

else if $\Sigma = \text{DataNN}$ **then**

$G \leftarrow \text{WeightedKNNGraph}(\mathbf{X}, k = 7)$

else

$G \leftarrow \text{DelaunayTriangulation}(\hat{\mathbf{Y}})$

end if

$\Gamma \leftarrow \hat{\mathbf{Y}}$ ▷ initial coordinates

 Run 5 iterations of Force-Directed Layout on Γ according to graph G

$\hat{\mathbf{Y}}_{\text{Post}} \leftarrow \Gamma$

return $\hat{\mathbf{Y}}_{\text{Post}}$

A.3.4 Additional experimental results

In Chapter 3, we perform aggregations over the considerable amount of experiment data collected in order to present comprehensive and interpretable visualizations. Here, we include additional visualizations of the same data to provide further details by peeling back levels of aggregation — *e.g.*, showing distributions of metric values directly instead of distribution of deltas in value — and using alternative visualization methods — *e.g.*, a box-plot instead of a ridgeline plot.

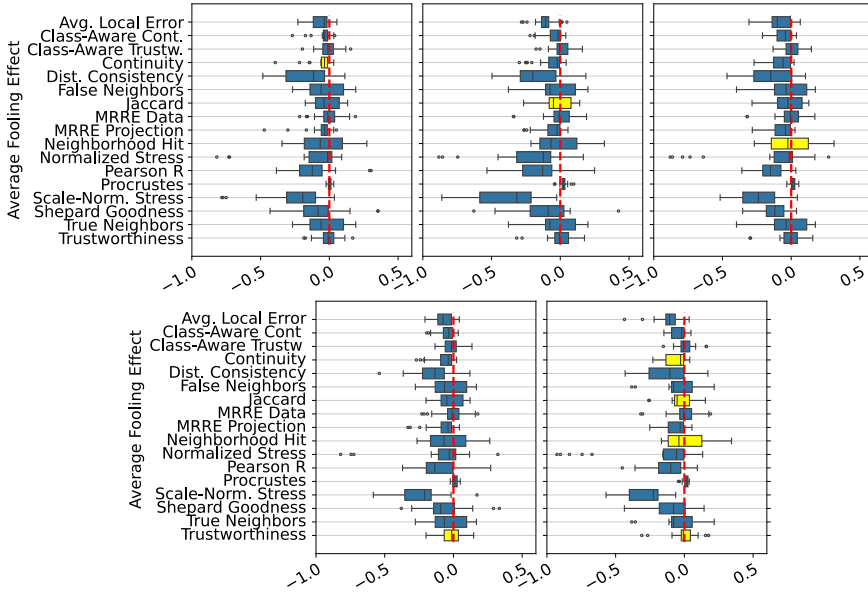


Figure A.11: Effect of fooling (as delta with respect to reference projection) grouped per target metric (from left to right, then top to bottom: Continuity, Jaccard, Neighborhood Hit, Trustworthiness, all previous metrics). All metrics have been remapped such that higher values are always better. We have highlighted the row(s) corresponding to the metric(s) used as fooling target. This is an alternative to Fig. 5.11a from the main text, using box-plots instead of a ridgeline plot.

A.3 FOOLING PROJECTION QUALITY METRICS



Figure A.12: Pre and post-fooling distributions of quality metric values, grouped by target fooling metric. In Figure 5.11, we compute the distribution of the variation in metrics $\Delta M = M^\dagger - M$. Here, we show the distributions of $M^\dagger$ and M separately. Metrics are re-mapped such that increases always mean quality improvements.

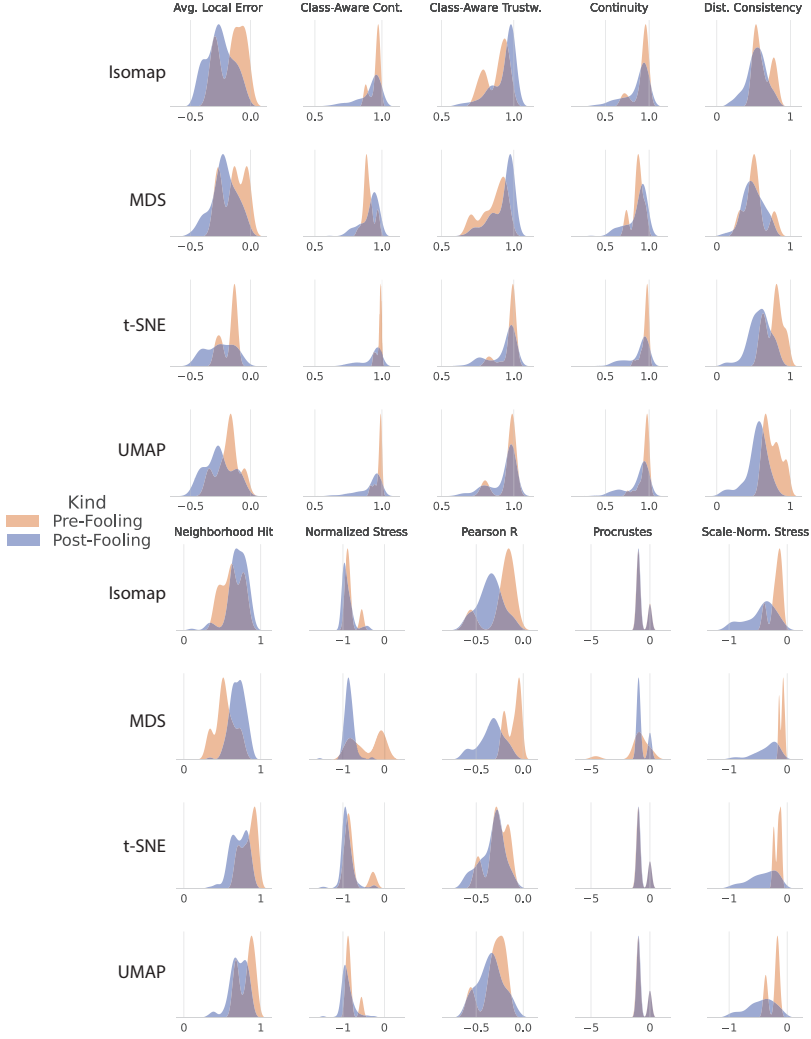


Figure A.13: Alternative visualization of data in Figure 5.11b in the main text. Distributions of pre- and post-fooling metrics, aggregated from all experiments, grouped by reference projection technique. In the main text, we compute the distribution of the variation in metrics $\Delta M = M^\dagger - M$. Here, we show the distributions of $M^\dagger$ and M separately. All metrics are re-mapped such that increases in metric values indicate improvements in quality.

A.3 FOOLING PROJECTION QUALITY METRICS

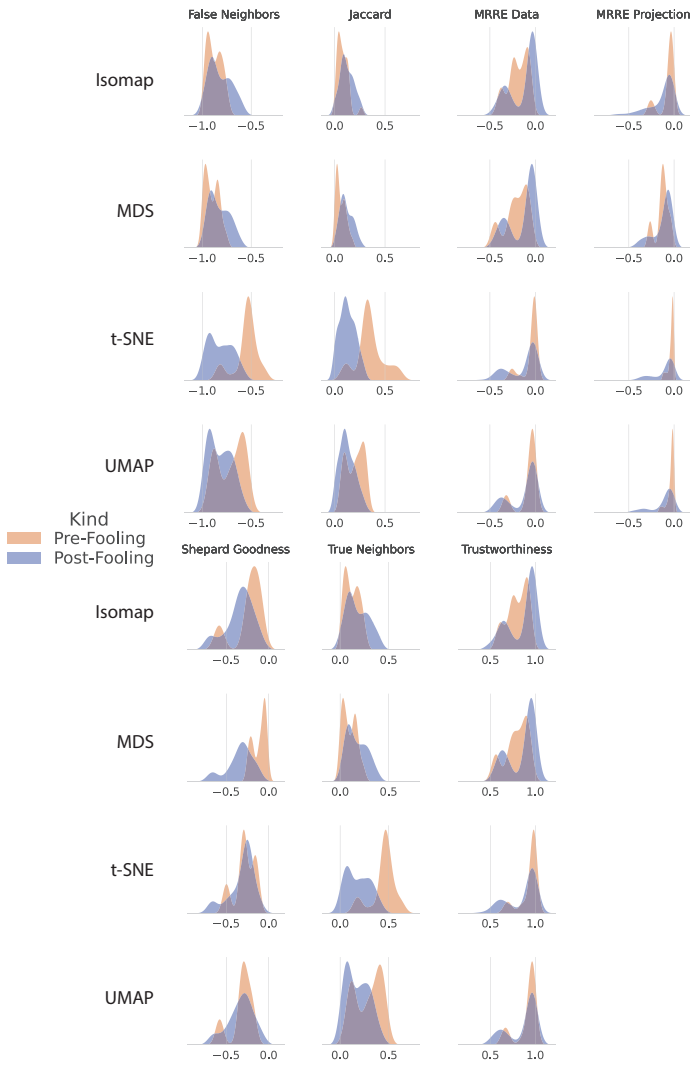


Figure A.14: Continuation of Figure A.13.

A.4 DIFFERENTIABLE DBMS

We include below additional results concerning the technique proposed in Chapter 6. We demonstrate the genericity of the ∂ DBM generation process by instantiating all our proposed views for distinct datasets — MNIST (LeCun et al., 1998), FashionMNIST (Xiao et al., 2017), and Quickdraw (Ha and Eck, 2018) —, classifiers — Feed-forward Neural Networks and Convolutional Neural Networks —, and projection algorithms — t-SNE (van der Maaten and Hinton, 2008), UMAP (McInnes and Healy, 2018), and Isomap (Tenenbaum et al., 2000).

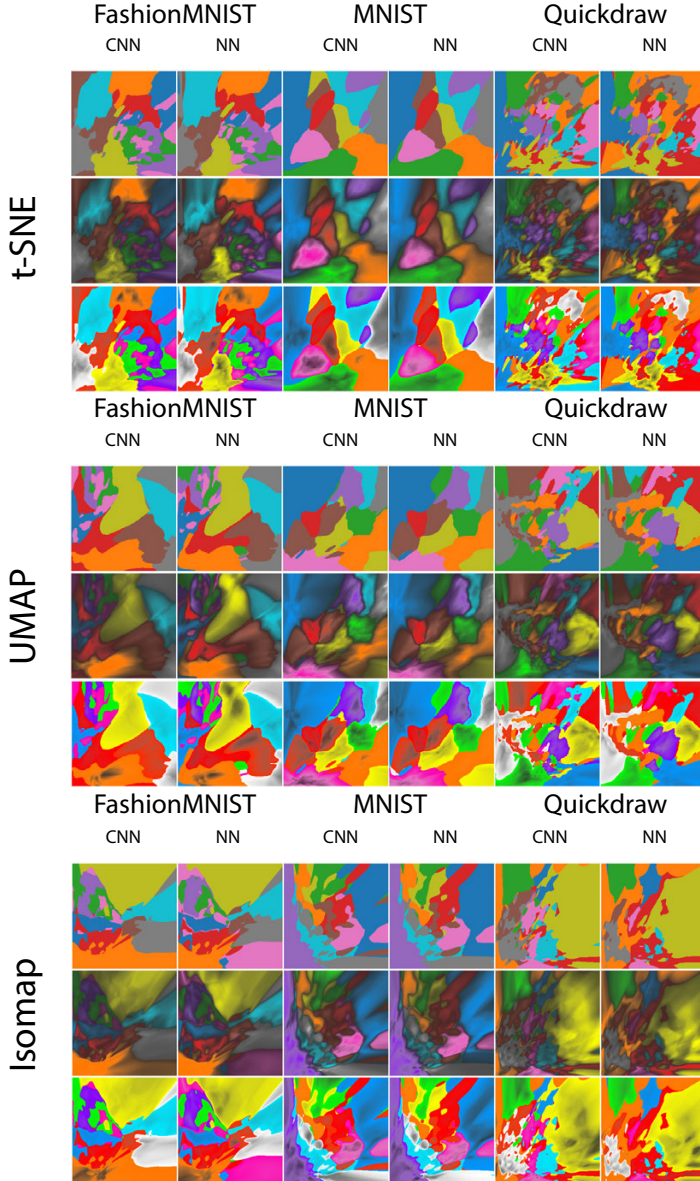
A.4.1 *Distance to adversarial example*

Figure A.15: DBMs (top), distance to decision boundary with safe regions highlighted (middle), and the same but with brittle regions highlighted (bottom), for all the classifier-projection-dataset combinations studied.

A.4.2 Distance to closest training data point

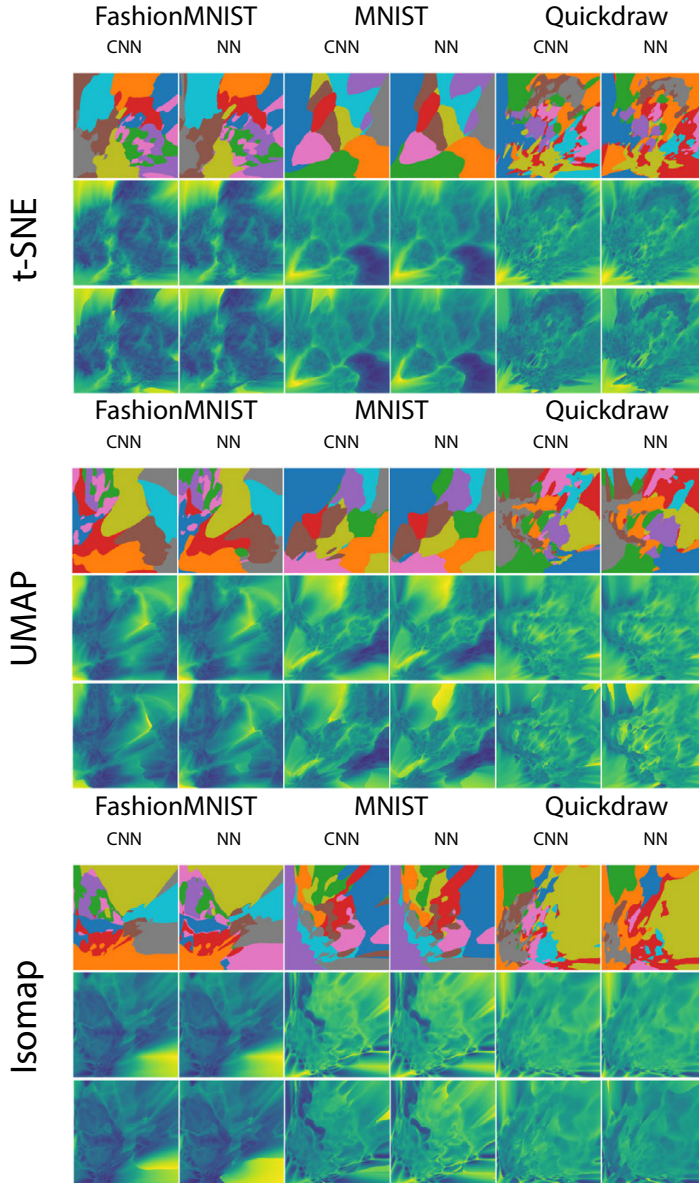


Figure A.16: DBMs for reference (top), distance to closest training sample views (middle = $\text{Dist}_{\text{train}}$, bottom = $\text{Dist}_{\text{SameClass}}$).

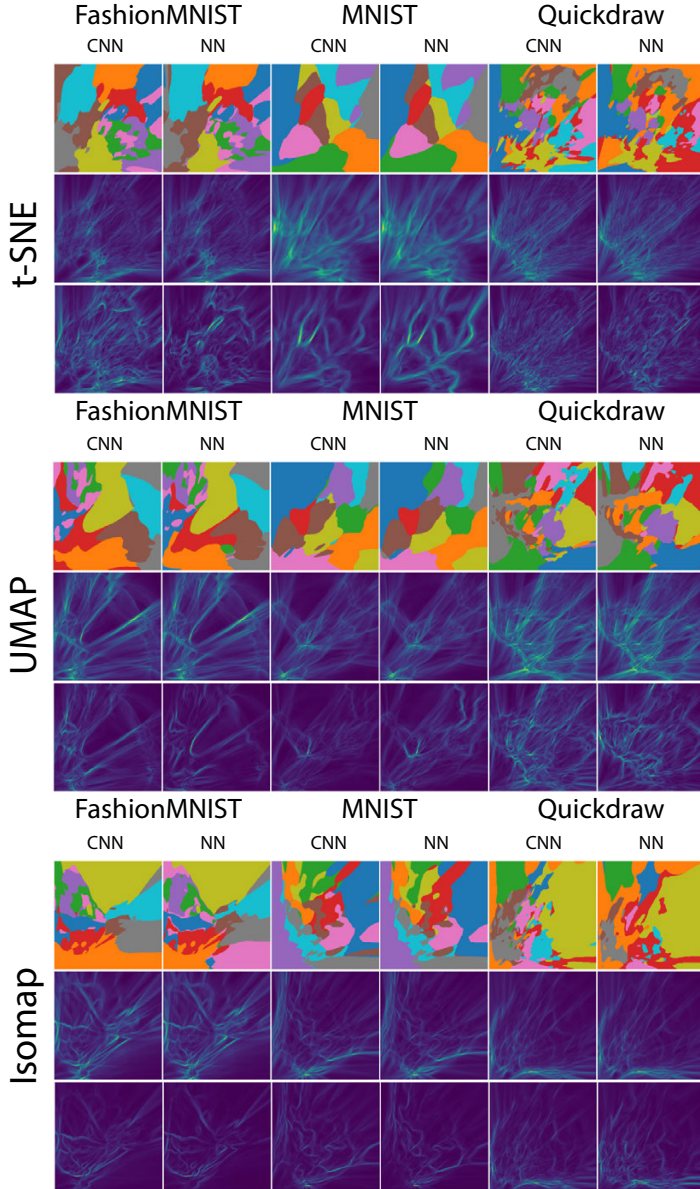
A.4.3 *Inverse projection space warping and class variability*

Figure A.17: DBMs for reference (top), SpaceExpansion view (middle), and ClassVariability (bottom).

A.4.4 *Sensitivity of training*

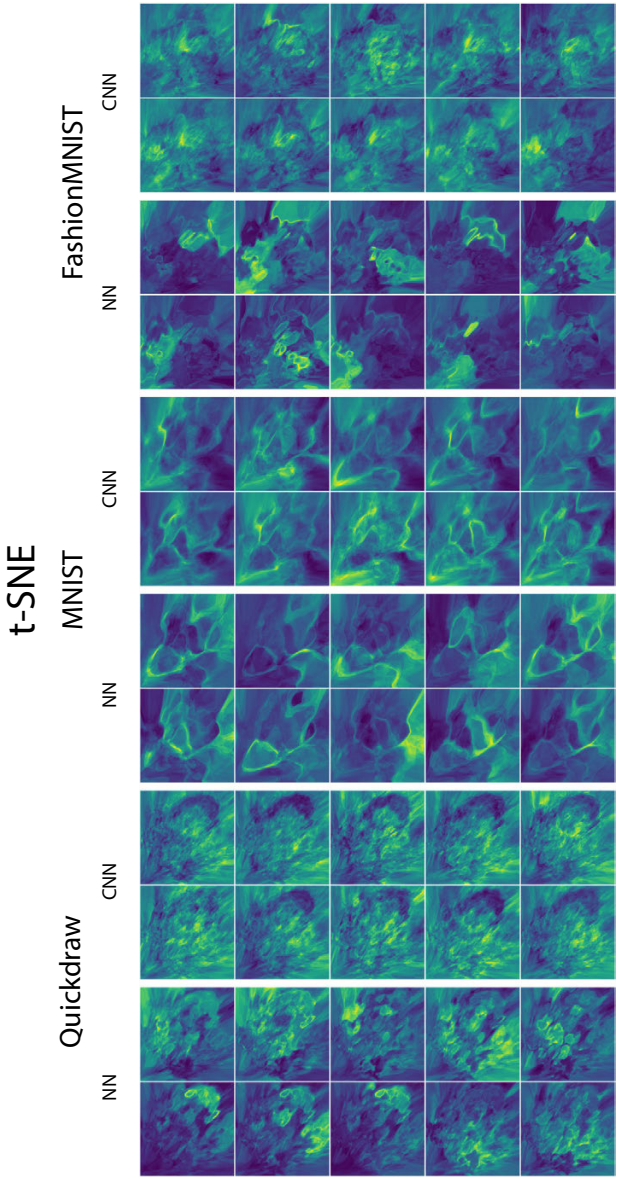


Figure A.18: Per-class training sensitivity (each group of 2 rows shows the values for classes 0-4 and then 5-9) for all studied combinations dataset and classifier, using t-SNE for projection.

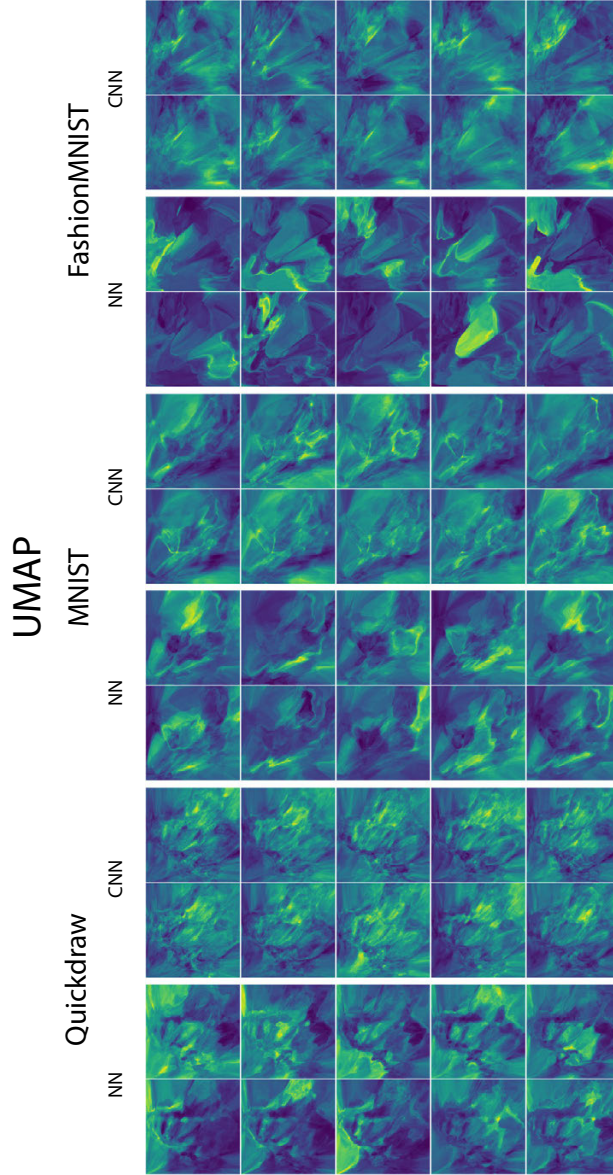


Figure A.19: Per-class training sensitivity (each group of 2 rows shows the values for classes 0-4 and then 5-9) for all studied combinations dataset and classifier, using UMAP for projection.

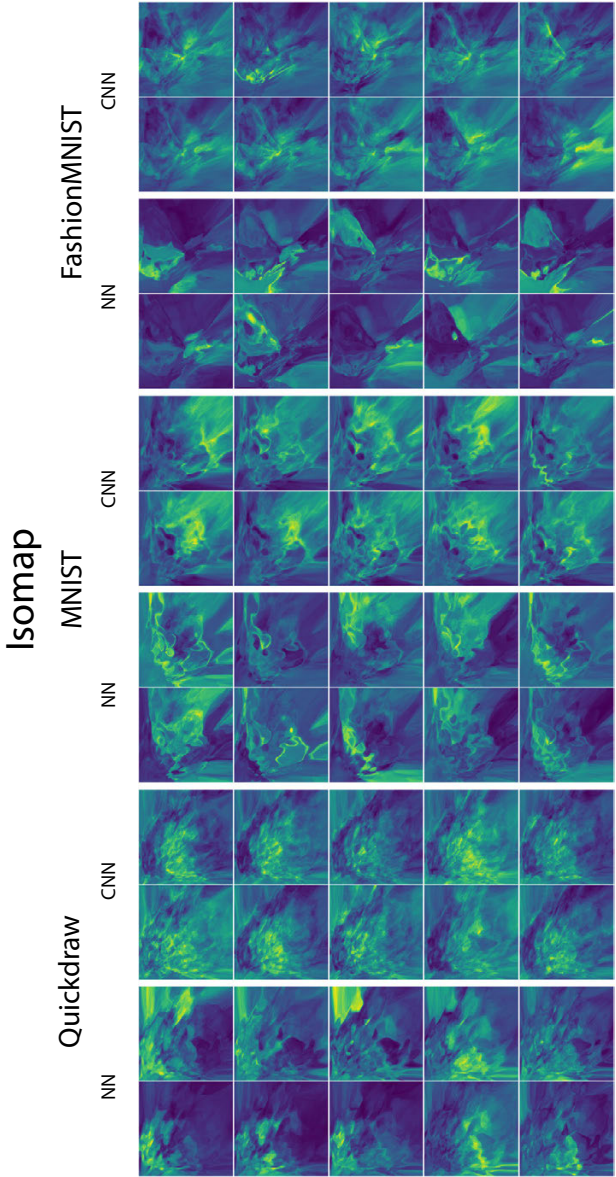


Figure A.20: Per-class training sensitivity (each group of 2 rows shows the values for classes 0-4 and then 5-9) for all studied combinations dataset and classifier, using Isomap for projection.

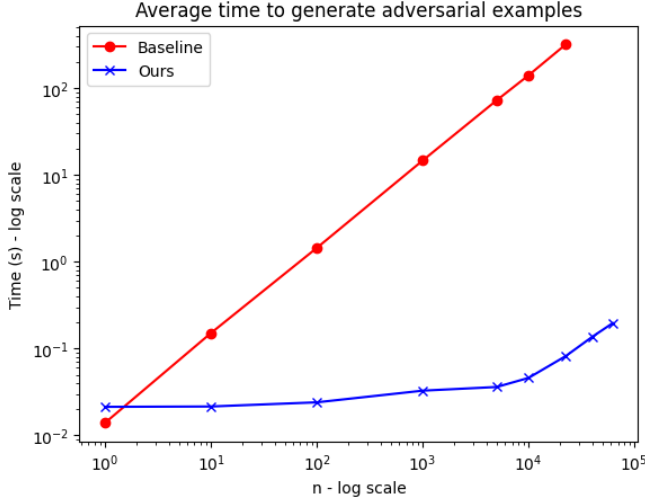


Figure A.21: Log-log plot of the time taken for the reference implementation of DeepFool to generate adversarial examples for increasing n compared to our batch-parallel implementation. Each measurement is an average over 3 runs and uses the same input in both versions.

A.4.5 Optimizing the implementation of DeepFool

DeepFool (Moosavi-Dezfooli et al., 2016) is our algorithm of choice for generating adversarial examples reliably. This is a key component of the Distance to Decision Boundary view. Seeing as we wish for our application to be interactive, the response times should be as low as possible, and a fast implementation of DeepFool is needed.

The reference implementation has been made publicly available by the authors at <https://github.com/LTS4/DeepFool>. However, it processes only *one* input point at a time. In ∂ DBMs, we are interested in generating high-resolution (300 by 300 pixels) views, therefore the cost of running the reference implementation in a loop becomes prohibitive.

We parallelize DeepFool, implementing the algorithm batch-wise, which incurs significant gains in speed, as can be seen in Fig. A.21. When operating on a single element, the overhead of the batch-parallel implementation is such that the reference implementation is faster than ours. The picture quickly changes though: our version of the algorithm takes only 44 milliseconds to generate adversarial examples for 5000 input points; comparing this to the reference implementation’s 59 *seconds* we see a dramatic speedup of four orders of magnitude.

BIBLIOGRAPHY

- M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, et al. TensorFlow: A system for large-scale machine learning. In *USENIX Symp. on Operating Systems Design and Implementation*, pages 265–283, 2016.
- M. M. Abbas, M. Aupetit, M. Sedlmair, and H. Bensmail. ClustMe: A visual quality measure for ranking monochrome scatterplots based on cluster patterns. *Computer Graphics Forum*, 38(3):225–236, 2019.
- M. M. Abbas, E. Ullah, A. Baggag, H. Bensmail, M. Sedlmair, and M. Aupetit. ClustML: A measure of cluster pattern complexity in scatterplots learnt from human-labeled groupings. *Information Visualization*, 23:105 – 122, 2021.
- G. Albuquerque, M. Eisemann, and M. Magnor. Perception-based visual quality measures. In *Proc. IEEE VAST*, pages 13–20, 2011.
- G. Alicioglu and B. Sun. A survey of visual analytics for explainable artificial intelligence methods. *Computers & Graphics*, 102:502–520, 2022.
- M. Allen and W. Yen. *Introduction to measurement theory*. Waveland Press, 1979.
- S. Amari. A theory of adaptive pattern classifiers. *IEEE TRANSACTIONS ON ELECTRONIC COMPUTERS*, 16(3):299, 1967.
- P. Angelov, N. Arnold, E. Soares, and P. Atkinson. Explainable artificial intelligence: an analytical review. *WIREs Data Mining Knowl Discov*, 11 (e1424), 2021.
- D. Anguita, A. Ghio, L. Oneto, X. Parra, and J. L. Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *Proc. IWAAL*, volume 7657 of *Lecture Notes in Computer Science*, pages 216–223. Springer, 2012.
- F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27: 17–21, 1973.
- G. Appleby, M. Espadoto, R. Chen, S. Goree, A. C. Telea, E. W. Anderson, and R. Chang. HyperNP: Interactive visual exploration of multidimensional projection hyperparameters. *Computer Graphic Forum*, 4(3):169–181, 2022.
- M. Aupetit and M. Sedlmair. SepMe: 2002 new visual separation measures. In *Proc. IEEE PacificVis*, 2016.

- M. Aupetit. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing*, 70(7-9):1304–1330, 2007.
- M. Barreno, B. Nelson, A. D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010. ISSN 0885-6125, 1573-0565.
- B. Barz and J. Denzler. Do We Train on Test Data? Purging CIFAR of Near-Duplicates. *Journal of Imaging*, 6(6):41, 2020. ISSN 2313-433X.
- M. Behrisch, M. Blumenschein, N. W. Kim, L. Shao, M. El-Assady, J. Fuchs, D. Seebacher, A. Diehl, U. Brandes, H. Pfister, T. Schreck, D. Weiskopf, and D. A. Keim. Quality metrics for information visualization. *Computer graphics forum*, 37(3):625–662, 2018.
- B. Benato, A. Falcão, and A. Telea. Linking data separation, visual separation, classifier performance using multidimensional projections. In *Springer CCIS 2103*, pages 229–255, 2024a.
- B. Benato, C. Grosu, A. X. Falcão, and A. Telea. Human-in-the-loop: Using classifier decision boundary maps to improve pseudo labels. *Computers & Graphics*, 124:104062, 2024b.
- B. C. Benato, A. X. Falcão, and A. C. Telea. Measuring the quality of projections of high-dimensional labeled data. *Computers & Graphics*, 116: 287–297, 2023. ISSN 0097-8493.
- Y. Bengio, A. C. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8): 1798–1828, 2013.
- E. Bertini and G. Santucci. Visual quality metrics. In *Proc. AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*, 2006.
- G. Blasilli, D. Kerrigan, E. Bertini, and G. Santucci. Towards a visual perception-based analysis of clustering quality metrics. In *Proc. VDM*, 2024.
- D. Blumberg, Y. Wang, A. Telea, D. Keim, and F. Dennig. MultiInv: Inverting multidimensional scaling projections and computing decision maps by multilateration. *Computers & Graphics*, 129:104234, 2025.
- C. Bredius, Z. Tian, and A. Telea. Visual exploration of neural network projection stability. In *Proc. MLVis*, 2022.
- K. Bunte, M. Biehl, and B. Hammer. A general framework for dimensionality reducing data visualization mapping. *Neural Computation*, 24(3): 771–804, 2012.

- R. J. G. B. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2013.
- T. T. Cao, K. Tang, A. Mohamed, and T. S. Tan. Parallel banding algorithm to compute exact distance transform with the GPU. In *Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 83–90, 2010.
- M. A. Carreira-Perpinán. A review of dimension reduction techniques. *Department of Computer Science. University of Sheffield. Tech. Rep. CS-96-09*, 9: 1–69, 1997.
- W. Castelein, Z. Tian, T. Mchedlidze, and A. Telea. Viewpoint-based quality for analyzing and exploring 3D multidimensional projections. In *Proc. IVAPP*, 2023.
- A. Chattopadhyay, A. Sarkar, P. Howlader, and V. N. Balasubramanian. Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks. In *2018 IEEE Winter Conference on Applications of Computer Vision, WACV 2018, Lake Tahoe, NV, USA, March 12-15, 2018*, pages 839–847. IEEE Computer Society, 2018.
- F. Chollet et al. Keras. <https://keras.io>, 2015.
- J. Choo and S. Lu. Visual analytics for explainable deep learning. *IEEE Computer Graphics and Applications*, 38(4):84–92, 2018.
- P. M. Ciarelli and E. Oliveira. Agglomeration and elimination of terms for dimensionality reduction. In *Ninth International Conference on Intelligent Systems Design and Applications, ISDA 2009, Pisa, Italy, November 30-December 2, 2009*, pages 547–552. IEEE Computer Society, 2009.
- A. E. Cinà, K. Grosse, A. Demontis, B. Biggio, F. Roli, and M. Pelillo. Machine learning security against data poisoning: Are we there yet? *CoRR*, abs/2204.05986, 2022.
- A. E. Cinà, K. Grosse, A. Demontis, S. Vascon, W. Zellinger, B. A. Moser, A. Oprea, B. Biggio, M. Pelillo, and F. Roli. Wild patterns reloaded: A survey of machine learning security against training data poisoning. *ACM Comput. Surv.*, 55(13s), 2023. ISSN 0360-0300.
- D. Coimbra, R. Martins, T. Neves, A. Telea, and F. Paulovich. Explaining three-dimensional dimensionality reduction plots. *Information Visualization*, 15(2):154–172, 2016.
- B. Colange, J. Peltonen, M. Aupetit, D. Dutykh, and S. Lespinats. Steering distortions to preserve classes and neighbors in supervised dimensionality reduction. In *Proc. NeurIPS*, pages 214–225, 2020.

- D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE TPAMI*, 24(5):603–619, 2002.
- W. Cui. Visual analytics: A comprehensive overview. *IEEE Access*, 7:81555–81573, 2019.
- J. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *JMLR*, 16:2859–2900, 2015.
- R. Cutura, S. Holzer, M. Aupetit, and M. Sedlmair. Viscoder: A tool for visually comparing dimensionality reduction algorithms. In *Proc. ESANN*, 2018.
- R. Cutura, C. Morariu, Z. Cheng, Y. Wang, D. Weiskopf, and M. Sedlmair. Hagrid: using Hilbert and Gosper curves to gridify scatterplots. *Journal of Visualization*, 2022.
- T. N. Dang and L. Wilkinson. ScagExplorer: Exploring scatterplots by their scagnostics. In *Proc. IEEE PacificVis*, pages 73–80, 2014.
- T. Davidson, D. Warmesley, M. W. Macy, and I. Weber. Automated hate speech detection and the problem of offensive language. In *Proceedings of the Eleventh International Conference on Web and Social Media, ICWSM 2017, Montréal, Québec, Canada, May 15-18, 2017*, pages 512–515. AAAI Press, 2017.
- T. R. Davidson, L. Falorsi, N. D. Cao, T. Kipf, and J. M. Tomczak. Hyper-spherical variational auto-encoders. In *Proceedings of the Thirty-Fourth Conference on Uncertainty in Artificial Intelligence, UAI 2018, Monterey, California, USA, August 6-10, 2018*, pages 856–865. AUAI Press, 2018.
- V. De Silva and J. B. Tenenbaum. Sparse multidimensional scaling using landmark points. Technical report, Stanford University, 2004.
- J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. Alemi, M. Hoffman, and R. A. Saurous. Tensorflow distributions, 2017.
- G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4): 12–25, 2015.
- H. Doraiswamy, J. Tierny, P. J. S. Silva, L. G. Nonato, and C. Silva. TopoMap: A 0-dimensional homology preserving projection of high-dimensional data. *IEEE TVCG*, 27(2):561–571, 2021.
- E. P. dos Santos Amorim, E. V. Brazil, J. Daniels, P. Joia, L. G. Nonato, and M. C. Sousa. iLAMP: Exploring high-dimensional spacing through backward multidimensional projection. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 53–62, 2012.

- N. Elmqvist, P. Dragicevic, and J.D. Fekete. Rolling the dice: Multidimensional visual exploration using scatterplot matrix navigation. *IEEE TVCG*, 14(6):1141–1148, 2008.
- J. Elson, J.R. Douceur, J. Howell, and J. Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. *CCS*, 7(366-374):15, 2007.
- S. van den Elzen, G. Andrienko, N. Andrienko, B. Fisher, R. Martins, J. Peltonen, A. Telea, and M. Verleysen. The flow of trust: A visualization framework for externalizing, exploring and explaining trust in ml applications. *IEEE CG & A*, 43(2):78–88, 2023.
- M. Espadoto, N. Hirata, and A. Telea. Deep learning multidimensional projections. *Information Visualization*, 19:247 – 269, 2019a.
- M. Espadoto, F.C.M. Rodrigues, N.S.T. Hirata, and R. Hirata. Deep learning inverse multidimensional projections. In *Proc. EuroVA*, page 5, 2019b.
- M. Espadoto, A. Falcao, N. Hirata, and A. Telea. Improving neural network-based multidimensional projections. In *Proc. IVAPP*. SciTePress, 2020.
- M. Espadoto, G. Appleby, A. Suh, D. Cashman, M. Li, C. Scheidegger, E. Anderson, R. Chang, and A. Telea. UnProjection: Leveraging inverse-projections for visual analytics of high-dimensional data. *IEEE Transactions on Visualization and Computer Graphics*, 29(2):1559–1572, 2023a.
- M. Espadoto, F. Rodrigues, N. Hirata, and A. Telea. Visualizing high-dimensional functions with dense maps. *SN Computer Science*, 4(230), 2023b.
- M. Espadoto, N.S.T. Hirata, and A.C. Telea. Self-supervised dimensionality reduction with neural networks and pseudo-labeling. In *Proc. IVAPP (2021)*, pages 27–37. SCITEPRESS, 2021a.
- M. Espadoto, R.M. Martins, A. Kerren, N.S.T. Hirata, and A.C. Telea. Toward a quantitative survey of dimension reduction techniques. *IEEE Transactions on Visualization and Computer Graphics*, 27(3):2153–2173, 2021b.
- R. Etemadpour, R. Motta, J.G. de Souza Paiva, R. Minghim, M.C.F. de Oliveira, and L. Linsen. Perception-based evaluation of projection methods for multidimensional data visualization. *IEEE TVCG*, 21(1):81–94, 2015.
- T.M.J. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Software: Practice and Experience*, 21, 1991.

- X. Geng, D. C. Zhan, and Z. H. Zhou. Supervised nonlinear dimensionality reduction for visualization and classification. *IEEE Trans. on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 35:1098–1107, 2005.
- Y. Goldberg and Y. Ritov. Local procrustes for manifold embedding: a measure of embedding quality and embedding algorithms. *Mach. Learn.*, 77(1):1–25, 2009.
- J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov. Neighbourhood components analysis. In *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.
- D. Ha and D. Eck. A neural representation of sketch drawings. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.
- D. J. Hand. Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Mach. Learn.*, 77(1):103–123, 2009.
- I. Hartskeerl, T. Mchedlidze, S. van Wageningen, P. Vangorp, and A. Telea. NNP-NET: Accelerating t-SNE Graph Drawing for Very Large Graphs by Neural Networks . In *Proc. Graph Drawing*, 2025.
- K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.
- D. Hendrycks and K. Gimpel. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *CoRR*, abs/1606.08415, 2016.
- N. Heulot, J. D. Fekete, and M. Aupetit. Visualizing dimensionality reduction artifacts: An evaluation, 2017. arXiv:1705.05283v1 [cs.HC].
- G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.
- P. Hoffman and G. Grinstein. A survey of visualizations for high-dimensional data mining. *Information Visualization in Data Mining and Knowledge Discovery*, 104:47–82, 2002.
- M. Hopkins, E. Reeber, G. Forman, and J. Suermondt. Spam-base. UCI Machine Learning Repository, 1999. DOI: <https://doi.org/10.24432/C53G6X>.
- K. Hornik, M. B. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

- P.J. Huber. Projection pursuit. *Ann. Statist.*, 13(2):435–475, 1985.
- J. J. Hull. A database for handwritten text recognition research. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(5):550–554, 1994.
- A. Inselberg. Parallel coordinates. In L. Liu and M. T. Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.
- P. Jaccard. Distribution de la flore alpine dans le bassin des dranses et dans quelques régions voisines. *Bulletin de la Societe Vaudoise des Sciences Naturelles*, 37:241–72, 1901.
- H. Jeon, A. Cho, J. Jang, S. Lee, J. Hyun, H. K. Ko, J. Jo, and J. Seo. ZADU: a Python library for evaluating the reliability of dimensionality reduction embeddings. *IEEE Visualization and Visual Analytics*, pages 196–200, 2023.
- H. Jeon, Y. H. Kuo, M. Aupetit, K. L. Ma, and J. Seo. Classes are not clusters: Improving label-based evaluation of dimensionality reduction. *IEEE TVCG*, 30(1):781–791, 2024a.
- H. Jeon, G. J. Quadri, H. Lee, P. Rosen, D. A. Szafir, and J. Seo. CLAMS: a cluster ambiguity measure for estimating perceptual variability in visual clustering. *IEEE TVCG*, 30(1):770–780, 2024b.
- S. Johansson and J. Johansson. Interactive dimensionality reduction through user-defined combinations of quality metrics. *IEEE TVCG*, 15, 2009.
- P. Joia, F. V. Paulovich, D. Coimbra, J. A. Cuminato, and L. G. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011.
- I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.
- L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.
- D. Keim, G. Andrienko, J. D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon. Visual analytics: Definition, process, and challenges. In *Information Visualization – Human-Centered Issues and Perspectives*, pages 154–175. Springer, 2008.
- Y. Kim, M. Espadoto, S. Trager, J. Roerdink, and A. Telea. SDR-NNP: Sharpened dimensionality reduction with neural networks. In *Proc. IVAPP*, 2022a.
- Y. Kim, A. Telea, S. Trager, and J. Roerdink. Visual cluster separation using high-dimensional sharpened dimensionality reduction. *Information Visualization*, 21(3):197–219, 2022b.

- D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. ICLR*, 2015.
- D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *2nd International Conference on Learning Representations, ICLR 2014*, 2014.
- D. Kotzias, M. Denil, N. de Freitas, and P. Smyth. From group to individual labels using deep features. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 597–606. ACM, 2015.
- A. Krizhevsky. Learning multiple layers of features from tiny images. *Computer Science Department University of Toronto Tech. Rep.*, pages 32–33, 2009.
- J. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.
- Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.
- J. Lee and M. Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72:1431–1443, 2009.
- S. Lespinats and M. Aupetit. CheckViz: Sanity check and topological clues for linear and non-linear mappings. *Computer Graphics Forum*, 30(1):113–125, 2011.
- S. Liu, D. Maljovec, B. Wang, P. T. Bremer, and V. Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE TVCG*, 23(3):1249–1268, 2015.
- Y. Liu, Z. Li, H. Xiong, X. Gao, and J. Wu. Understanding of internal clustering validation measures. In *ICDM*, pages 911–916. IEEE Computer Society, 2010.
- S. P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.
- J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: A review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.
- A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA*, pages 142–150. The Association for Computer Linguistics, 2011.

- L. van der Maaten and G. E. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.
- L. van der Maaten. Learning a parametric embedding by preserving local structure. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 384–391. PMLR, 2009.
- A. Machado and A. Telea. APPA: A cluster-preserving approximating parametric projection algorithm. In *Proc. GRIVAPP*. SciTePress, 2026.
- A. Machado, A. Telea, and M. Behrisch. ShaRP: Shape-regularized multi-dimensional projections. In *Proc. EuroVA*, 2023.
- A. Machado, M. Behrisch, and A. Telea. Projection fooler source code, 2024a. <https://github.com/amreis/fooling-projection-metrics>.
- A. Machado, M. Behrisch, and A. Telea. Online experimental results of fooling projection quality metrics, 2024b. <https://amreis.github.io/fool-proj-metrics/>.
- A. Machado, M. Behrisch, and A. Telea. Accelerated projection quality metrics, 2024c. <https://pypi.org/p/tensorflow-projection-qm/>.
- A. Machado, M. Behrisch, and A. Telea. Extensible TensorFlow implementations of projection quality metrics. In *Proc. VisGap*. Eurographics, 2025a.
- A. Machado, M. Behrisch, and A. Telea. Necessary but not sufficient: Limitations of projection quality metrics. *Computer Graphics Forum*, 44(3), 2025b.
- A. Machado and A. Telea. APPA implementation, 2025. <https://github.com/amreis/appa>.
- A. Machado, M. Behrisch, and A. Telea. Exploring classifiers with differentiable decision boundary maps. *Computer Graphics Forum*, 43(3):e15109, 2024d. ISSN 1467-8659.
- A. Machado, A. Telea, and M. Behrisch. Controlling the scatterplot shapes of 2D and 3D multidimensional projections. *Computers & Graphics*, 2024e.
- A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015.
- R. Martins, R. Minghim, and A. Telea. Explaining neighborhood preservation for multidimensional projections. In *Proc. CGVC*. Eurographics, 2015.

- R. M. Martins, D. B. Coimbra, R. Minghim, and A. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014. ISSN 0097-8493.
- J. Matejka and G. Fitzmaurice. Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing. *Proc. ACM CHI*, 2017.
- L. McInnes and J. Healy. UMAP: Uniform manifold approximation and projection for dimension reduction. *ArXiv*, abs/1802.03426, 2018.
- A. Meijster, J. Roerdink, and W. Hesselink. A general algorithm for computing distance transforms in linear time. In *Mathematical Morphology and its Applications to Image and Signal Processing*, pages 331–340. Springer, 2002.
- T. Modrakowski, M. Espadoto, A. Falcão, N. Hirata, and A. Telea. Improving deep learning projections by neighborhood analysis. In *Computer Vision, Imaging and Computer Graphics Theory and Applications*, pages 127–152. Springer, 2022.
- S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: A simple and accurate method to fool deep neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 2574–2582. IEEE Computer Society, 2016.
- C. Morariu, A. Bibal, R. Cutura, B. Frénay, and M. Sedlmair. Predicting user preferences of dimensionality reduction embedding quality. *IEEE TVCG*, 29(1):745–755, 2023.
- S. Moro, P. Cortez, and P. Rita. A data-driven approach to predict the success of bank telemarketing. *Decis. Support Syst.*, 62:22–31, 2014.
- J. T. Morton, J. Silverman, G. Tikhonov, H. Lähdesmäki, and R. Bonneau. Scalable estimation of microbial co-occurrence networks with variational autoencoders. *bioRxiv*, 2021.
- R. Motta, R. Minghim, A. de Andrade Lopes, and M. C. F. de Oliveira. Graph-based measures to assist user assessment of multidimensional projections. *Neurocomputing*, 150:583–598, 2015.
- T. Munzner. *Visualization Analysis and Design: Principles, Techniques, and Practice*. CRC Press, 2014.
- S. A. Nene, S. K. Nayar, and H. Murase. Columbia object image library (coil-20). Technical Report CUCS-005-96, Department of Computer Science, Columbia University, 1996.

- Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.
- L. Nonato and M. Aupetit. Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG*, 2018.
- A. Oliveira, M. Espadoto, R. Hirata, N. Hirata, and A. Telea. Improving self-supervised dimensionality reduction: Exploring hyperparameters and pseudo-labeling strategies. In *Springer CCIS 1691*, pages 135–161, 2023a.
- A. Oliveira, M. Espadoto, R. H. Jr, and A. Telea. Stability analysis of supervised decision boundary maps. *SN Computer Science*, 4(226), 2023b.
- A. A. Oliveira, M. Espadoto, R. Hirata Jr, and A. C. Telea. SDBM: Supervised Decision Boundary Maps for Machine Learning Classifiers. In *VISIGRAPP (3: IVAPP)*, pages 77–87, 2022.
- D. Pan, X. Li, and D. Zhu. Explaining deep neural network models with adversarial gradient integration. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, Virtual Event / Montreal, Canada, 19-27 August 2021*, pages 2876–2883. ijcai.org, 2021.
- A. V. Pandey, J. Krause, C. Felix, J. Boy, and E. Bertini. Towards understanding human similarity perception in the analysis of large sets of scatter plots. In *Proc. ACM CHI*, pages 3659–3669, 2016.
- F. V. Paulovich, L. G. Nonato, R. Minghim, and H. Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG*, 14(3):564–575, 2008.
- K. Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.
- F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- T. Plötz and S. Roth. Neural nearest neighbors networks. In *Proc. NeurIPS*, pages 1095–1106, 2018.
- J. Poco, R. Etemadpour, and F. Paulovich. A framework for exploring multidimensional data with 3D projections. *Comput Graph Forum*, 30(3):1111–1120, 2011.

- R. Rao and S.K. Card. The table lens: Merging graphical and symbolic representations in an interactive focus+ context visualization for tabular information. In *Proc. ACM CHI*, 1994.
- P. Rauber, A. Falcao, and A. Telea. Projections as visual aids for classification system design. *Information Visualization*, 17(4):282–305, 2017.
- R. A. Rensink and G. Baldridge. The perception of correlation in scatterplots. *Computer Graphics Forum*, 29(3):1203–1210, 2010.
- F. C. M. Rodrigues, M. Espadoto, R. H. Jr, and A. Telea. Constructing and visualizing high-quality classifier decision boundary maps. *Information*, 10(9):280–297, 2019.
- F. C. M. Rodrigues, R. Hirata, and A. C. Telea. Image-based visualization of classifier decision boundaries. In *31st SIBGRAPI Conference on Graphics, Patterns and Images, SIBGRAPI 2018, Paraná, Brazil, October 29 - Nov. 1, 2018*, pages 353–360. IEEE Computer Society, 2018.
- B. L. Rosa, G. Blasilli, R. Bourqui, D. Auber, G. Santucci, R. Capobianco, E. Bertini, R. Giot, and M. Angelini. State of the art of visual analytics for eXplainable deep learning. *Computer Graphics Forum*, 42(1):319–355, 2023.
- F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- M. Rosenblatt. Remarks on some nonparametric estimates of a density function. *The Annals of Mathematical Statistics*, 27(3):832–837, 1956.
- W. Samek, G. Montavon, A. Vedaldi, L.K. Hansen, and K.R. Müller. *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer LNCS, 2019.
- T. Schreck, T. von Landesberger, and S. Bremm. Techniques for precision-based visual analysis of projected data. *Inf. Vis.*, 9(3):181–193, 2010.
- A. Schulz, A. Gisbrecht, and B. Hammer. Using discriminative dimensionality reduction to visualize classifiers. *Neural Processing Letters*, 42:27–54, 2015.
- A. Schulz, F. Hinder, and B. Hammer. DeepView: Visualizing Classification Boundaries of Deep Neural Networks as Scatter Plots Using Discriminative Dimensionality Reduction. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 2305–2311, 2020.
- M. Sedlmair, A. Tatu, T. Munzner, and M. Tory. A taxonomy of visual cluster separation factors. *Computer Graphics Forum*, 31(3):1335–1344, 2012.

- M. Sedlmair and M. Aupetit. Data-driven evaluation of visual quality measures. *Computer Graphics Forum*, 34(3):201–210, 2015.
- M. Sedlmair, T. Munzner, and M. Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE Trans. Vis. Comput. Graph.*, 19(12):2634–2643, 2013.
- L. Shao, T. Schleicher, M. Behrisch, T. Schreck, I. Sipiran, and D. Keim. Guiding the exploration of scatter plot data using motif-based interest measures. *J. Vis. Lang. Comput.*, 36:1–12, 2016.
- L. Sharan, R. Rosenholtz, and E. Adelson. Material perception: What can you see in a brief glance? *Journal of Vision - J VISION*, 9:784–784, 2010.
- S. Siegel and N. J. Castellan. *Nonparametric statistics for the behavioral sciences*. McGraw Hill, 1988.
- R. da Silva, P. Rauber, and A. Telea. Beyond the third dimension: Visualizing high-dimensional data with projections. *IEEE Computing in Science & Engineering*, 18(5):98–107, 2016.
- M. Sips, B. Neubert, J. P. Lewis, and P. Hanrahan. Selecting good views of high-dimensional data using class consistency. *Computer graphics forum*, 28(3):831–838, 2009.
- K. Smelser, J. Miller, and S. Kobourov. “Normalized Stress” is Not Normalized: How to Interpret Stress Correctly . In *Proc. IEEE BELIV*, pages 41–50. IEEE Computer Society, 2024.
- C. Sorzano, J. Vargas, and A. Pascual-Montano. A survey of dimensionality reduction techniques, 2014. arXiv:1403.2877 [stat.ML].
- N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.
- J. Steinhardt, P. W. W. Koh, and P. S. Liang. Certified Defenses for Data Poisoning Attacks. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- A. Tatu, P. Bak, E. Bertini, D. Keim, and J. Schneidewind. Visual quality metrics and human perception: An initial study on 2d projections of large multidimensional data. In *Proceedings of the International Conference on Advanced Visual Interfaces, AVI '10*, page 49–56. Association for Computing Machinery, 2010.
- A. Telea. Combining extended table lens and treemap techniques for visualizing tabular data. In *Proc. EuroVis*, pages 120–127, 2006.

- A. Telea, A. Machado, and Y. Wang. Seeing is learning in high dimensions: The synergy between dimensionality reduction and machine learning. In *SN Computer Science*, volume 5 number 3, page 279, 2024.
- A. C. Telea. *Data Visualization: Principles and Practice, Second Edition*. A. K. Peters, Ltd., 2nd edition, 2014.
- J. Tenenbaum, V. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290 5500:2319–23, 2000.
- M. Thoma. The Reuters dataset, 2017. URL <https://martin-thoma.com/nlp-reuters>.
- Z. Tian, X. Zhai, G. van Steenpaal, L. Yu, E. Dimara, M. Espadoto, and A. Telea. Quantitative and qualitative comparison of 2D and 3D projection techniques for high-dimensional data. *Information*, 12(239), 2021.
- W. S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17:401–419, 1952.
- J. W. Tukey and P. A. Tukey. *Computer graphics and exploratory data analysis: An introduction*, volume 5. CRC Press, 1988.
- L. van der Maaten and E. Postma. Dimensionality reduction: A comparative review. Technical report, Tilburg University, Netherlands, 2009.
- V. Vapnik. Principles of risk minimization for learning theory. In J. Moody, S. Hanson, and R. Lippmann, editors, *Advances in Neural Information Processing Systems*, volume 4. Morgan-Kaufmann, 1991.
- A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, et al. Attention is all you need. In *Proc. NeurIPS*, pages 5998–6008, 2017.
- J. Venna and S. Kaski. Local multidimensional scaling. *Neural Networks*, 19 (6):889–899, 2006. ISSN 0893-6080.
- E. Vernier, R. Garcia, I. da Silva, J. Comba, and A. Telea. Quantitative evaluation of time-dependent multidimensional projection techniques. *Computer Graphics Forum*, 39(3):241–252, 2020a.
- E. Vernier, M. Sondag, J. Comba, B. Speckmann, A. Telea, and K. Verbeek. Quantitative comparison of time-dependent treemaps. *Computer Graphics Forum*, 39(3):393–404, 2020b.
- V. M. Vu, A. Bibal, and B. Frénay. Constraint preserving score for automatic hyperparameter tuning of dimensionality reduction methods for visualization. *IEEE Trans. on Artificial Intelligence*, 2(3):269–282, 2021.

- B. Wang and K. Mueller. The subspace voyager: Exploring high-dimensional data along a continuum of salient 3D subspaces. *IEEE TVCG*, 24(2):1204–1222, 2017.
- Y. Wang and A. Telea. Fundamental limitations of inverse projections and decision maps. In *Proc. IVAPP*, 2024.
- Y. Wang, A. Machado, and A. Telea. Quantitative and qualitative comparison of decision map techniques for explaining classification models. *Informatics*, 16(9), 2023.
- Y. Wang, C. Grosu, and A. Telea. Computing fast and accurate maps for explaining classification models. *Computers & Graphics*, 129:104230, 2025.
- Y. Wang, K. Feng, X. Chu, J. Zhang, C. Fu, M. Sedlmair, X. Yu, and B. Chen. A perception-driven approach to supervised dimensionality reduction for visualization. *IEEE Trans. Vis. Comput. Graph.*, 24(5):1828–1840, 2018.
- Y. Wang, Z. Wang, T. Liu, M. Correll, Z. Cheng, O. Deussen, and M. Sedlmair. Improving the robustness of scagnostics. *IEEE Trans. Vis. Comput. Graph.*, 26(1):759–769, 2020.
- M. Wattenberg, V. Fernanda, and I. Johnson. How to Use t-SNE Effectively. *Distill*, 2016.
- J.J. van Wijk. Unfolding the Earth: Myriahedral projections. *The Cartographic Journal*, 45(1):32–42, 2008.
- L. Wilkinson, A. Anand, and R. Grossman. High-dimensional visual analytics: Interactive exploration guided by pairwise views of point distributions. *IEEE TVCG*, 12(6):1363–1372, 2006.
- L. Wilkinson, A. Anand, and R. L. Grossman. Graph-theoretic scagnostics. In *Proc. IEEE InfoVis*, pages 157–164, 2005.
- H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.
- A. Yates, A. Webb, M.F. Sharpnack, H. Chamberlin, K. Huang, and R. Machiraju. Visualizing multidimensional data with glyph SPLOMs. *Computer Graphics Forum*, 33(3):301–310, 2014.
- H. Yin. Nonlinear dimensionality reduction and data visualization: A review. *Intl. Journal of Automation and Computing*, 4(3):294–303, 2007.
- K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C.C. Loy. Domain generalization: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(4):4396–4415, 2023.

BIOGRAPHY

Álister Machado dos Reis was born in Porto Alegre, Brazil, on December 19th, 1995. He received his Bachelor's degree in Computer Science from the Federal University of Rio Grande do Sul (UFRGS, Brazil) in 2018. During these studies, he was admitted to an exchange program leading to a double degree with the National Polytechnic Institute of Grenoble (INP Grenoble, France), where he obtained his M. Sc. in Data Science (*cum laude*).

Following his undergraduate studies, Álister was employed as a Software Engineer at Google Brazil, in the city of Belo Horizonte, where he stayed for three years. During this time, he honed his software development and project management skills, which would later play a crucial role in his academic endeavors. However, the wish to dedicate himself to the scientific career was always present in the back of Álister's mind.

In 2021, he joined the Visualization and Graphics group (VIG) at Utrecht University in the Netherlands as a Ph.D. Candidate. Switching careers and moving abroad presented challenges, but the desire to have deeper scientific experiences — as well as a strong love of learning and teaching — provided excellent motivation to pursue this opportunity. Álister's natural interest in Machine Learning and statistical models such as Neural Networks found clear application in the field of Data Visualization, leading to his work on Neural Dimensionality Reduction.

Aside from his tendency to want to explore the deep mathematics of statistical modeling and Machine Learning, Álister is also interested in how to best teach students in a way that encourages participation and critical thinking. As such, he is an avid reader of pedagogy scholars such as Paulo Freire and bell hooks. He wishes to build knowledge and connections — among scientific subjects and people alike — in an era where the pressure for 'automating away' humanity is ever growing.

ACKNOWLEDGMENTS

I would like to begin by thanking my supervisor, professor Alexandru Telea, without whom this work would have certainly not been as fruitful or enjoyable. Thank you for extending me your hand and truly *collaborating* with me. I have, since day one, felt your trust in my potential and your guiding, but never overbearing, hand on my shoulder.

Many thanks also go to my second advisor, Dr. Michael Behrisch. Thank you for providing me the opportunity to develop myself across research, teaching, and supervision, thus contributing to my (ongoing) maturing as an academic.

I also extend my thanks to all members of the VIG group with whom I have shared my journey throughout these five years. You have provided support, feedback, and an environment which I felt happy to be part of. There is no better place to learn to be a researcher than with a group of people that is honest, welcoming, and genuinely curious about each other's results, opinions, and questions.

My next round of thanks goes to my family and friends which, despite the physical distance, never allowed me to feel alone. To my mother Cláurea whom I love, and her steadfast trust in my decisions and in my ability to achieve my goals. To my family members — of which I must name but a few — Fernanda, Érika, Pauline, Bruna, and Antônio, whose support is invaluable to me. To my loving friends, without whom I cannot imagine being able to complete this journey: Betina and Hermes, Eduardo and Agathe, Bruna, Marialena, Manos, and Bárbara.

Lastly, I wish to thank all of the mentors I have had along the way and all of the teachers who dedicated themselves to sharing and building knowledge. You will always be sources of inspiration to me.

I firmly believe that nobody ever achieves anything alone, and completing this doctorate is no different. I consider myself extremely lucky to have received support and encouragement from so many incredible people along these years.

COLOPHON

This document was typeset using the typographical look-and-feel classicthesis developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography *"The Elements of Typographic Style"*. classicthesis is available for both L^AT_EX and L^yX:

<http://code.google.com/p/classicthesis/>

Final Version as of July 9, 2026 (classicthesis).