Learning Multidimensional Projections with Neural Networks

Espadoto, Mateus

DOI:
10.33612/diss.166005659

LEARNING MULTIDIMENSIONAL PROJECTIONS WITH NEURAL NETWORKS

MATEUS ESPADOTO

Cover: MNIST dataset projected using the Self-Supervised Network Projection technique and ground truth labels.

Learning Multidimensional Projections with Neural Networks

Mateus Espadoto
PhD Thesis

This thesis is the result of a joint PhD between the University of São Paulo and the University of Groningen.

university of groningen

Universidade de São Paulo

Learning Multidimensional Projections with Neural Networks

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. C. Wijmenga
and in accordance with
the decision by the College of Deans,
and
to obtain the degree of PhD at the
University of São Paulo
on the authority of the
Rector Magnificus Prof. V. Agopyan.

Double PhD Degree

This thesis will be defended in public on

Friday 16 April 2021 at 14.30 hours

by

Mateus Espadoto

born on July 17th, 1980
in São Paulo, Brazil

Supervisors
Prof. Alexandru Cristian Telea
Prof. Nina Sumiko Tomita Hirata

Assessment committee
Prof. Nicolai Petkov
Prof. Michael Biehl
Prof. Joao Luiz Dihl Comba
Prof. Roberto Marcondes Cesar Jr.

"We are here to laugh at the odds and live our lives so well that Death will tremble to take us."

— Charles Bukowski

ABSTRACT

Understanding data is essential in our modern world. There are many tools to help one understand data, and visualization tools occupy a prominent role in this spectrum. However, as we are flooded with ever more complicated data, the challenges and demands posed on visualization tools keep increasing.

This thesis looks at the particular case of multidimensional data, which consists of (large) collections of observations, each described by tens up to thousands of independent measurements, or dimensions. Multidimensional projection techniques, also known as dimensionality reduction techniques, are a key visualization tool for understanding such data. They aim to present the high-dimensional data by means of 2D or 3D scatterplots, and help data analysts uncover patterns in the data which would not be possible otherwise. In particular, projections provide the ability of visualizing how data *observations* relate to each other in a dataset. This can have many uses, such as identifying if and how observations are grouped, how close those data groups are in data space, and if those groups are tight or spread out, to name just a few. Over a span of more than a hundred years many projection techniques were developed, with new techniques currently under development. However, the problem of dimensionality reduction is far from solved – there is no 'perfect' projection method out there. Also, the related problem of evaluating projections is unsolved, since there are no single, clear-cut criteria, that state unequivocally what is a good projection.

In this thesis, we extend the current state-of-the-art on dimensionality reduction along three main directions, as follows.

First, we provide a quantitative benchmark to assess the quality of projection methods. Our benchmark surpasses all similar attempts we are aware of to date in terms of the number of evaluated techniques, tested datasets, quality metrics used, and parameter combinations studied. This provides us with a clear landscape of the area, and brings important insights over how good are the existing techniques and evaluation metrics. Our benchmark also offers a detailed, concrete, and quantitative methodology for practitioners to choose optimal projections based on the concerns they want to satisfy in practice.

Secondly, we revisit a research area that has not developed much since Self-Organizing Maps emerged in the 1980's: The use of neural networks for multidimensional projections. We show how neural networks can be used to create projections that surpass existing techniques in terms of scalability, out-of-sample capability, cluster separation, inverse mapping, and ease of use.

Last but not least, we propose several novel applications of multidimensional projections that highlight their importance as tools for high-dimensional data analysis, with a focus on understanding and improving the behavior of classifier models and optimization problem solvers.

SAMENVATTING

Het begrijpen van gegevens is essentieel in onze moderne wereld. Visualisatiegereedschap neemt een prominente plaats in het spectrum van instrumenten die dit doel dienen. Als het vloed van gegevens van toenemende complexiteit groeit, wordt visualisatiegereedschap geconfronteerd met steeds toenemende eisen.

Dit proefschrift onderzoekt het geval van multidimensionale gegevens die bestaan uit (grote) verzamelingen observaties, elk beschreven door tientallen tot duizenden onafhankelijke metingen of dimensies. Multidimensionale projectietechnieken, ook bekend als dimensionaliteitsreductietechnieken, zijn een vooraanstaand visualisatietype bedoeld voor deze gegevens. Zij presenteren hoog-dimensionale gegevens als 2D of 3D *scatterplots* en helpen data-analysten in het detecteren van verborgen patronen in de data. Projecties helpen in het visualiseren van relaties tussen de *observaties* in de gegevens. Dit ondersteunt taken zoals vinden van groepen van gerelateerde observaties, schatten van afstanden tussen de groepen, en schatten van de spreiding binnen een groep. Vele projectietechnieken zijn ontwikkeld over de laatste eeuw; nieuwe technieken worden steeds aangeboden. Toch is het probleem van dimensionaliteitsreductie verre van opgelost – er is geen 'perfecte' dergelijke techniek. Bovendien het gerelateerd probleem van evaluatie van projecties is nog niet opgelost gezien het feit dat criteria voor een goede projectie zijn nog niet uitgekristalliseerd.

Dit proefschrift breidt uit de bestaande kennis over dimensionaliteitsreductie in drie richtingen, als volgt.

Ten eerste presenteren wij een *benchmark* voor het meten van de kwaliteit van projectiemethodes. Deze *benchmark* overtreft alle dergelijke initiatieven die ons bekend zijn in het aantal geëvalueerde technieken, geteste dataverzamelingen, en parametercombinaties under studie. De resultaten ervan geven een duidelijk beeld van het landschap van projecties evenals hun kwaliteit en die van de bestaande metrieken. De *benchmark* geeft ook een gedetailleerde, concrete, en kwantitatieve methodologie die gebruikers in staat stelt om optimale projecties te kiezen in lijn met hun praktische eisen.

Ten tweede analyseren wij opnieuw het gebruik van neurale netwerken voor projectiedoeleinden, een onderzoeksrichting dat niet veel verder ontwikkeld is sinds de komst van *Self-Organizing Maps* in de jaren '80 van de vorige eeuw. We laten zien hoe neurale netwerken projecties kunnen creëren beter dan bestaande technieken betreffend schaalbaarheid, *out-of-sample* eigenschappen, clusterscheiding, inverse mapping, en gebruikersgemak.

Ten slotte presenteren wij een aantal toepassingen van multidimensionale projecties die hun belang als instrumenten voor hoog-dimensionale data-analyse onderbouwt, met nadruk op het begrijpen en verbeteren van het gedrag van classificatiemodellen en oplossingsmethodes voor optimalisatieproblemen.

RESUMO

No mundo de hoje, compreender dados é essencial. Existem muitas ferramentas que nos auxiliam na compreensão dos dados, e as ferramentas de visualização ocupam um lugar proeminente neste espectro. No entanto, ao lidarmos com uma quantidade enorme de dados cada vez mais complicados, os desafios para as ferramentas de visualização são cada vez maiores.

Esta tese analisa o caso particular de dados multivariados, que consistem de grandes coleções de observações, cada uma descrita por dezenas ou até milhares de variáveis independentes ou dimensões. As técnicas de Projeções Multidimensionais, também conhecidas como técnicas de Redução de Dimensionalidade, são ferramentas chave para a visualização e compreensão de dados deste tipo. Estas técnicas buscam mostrar dados multivariados em gráficos 2D ou 3D, o que pode ajudar analistas de dados a descobrir padrões nos dados que não seriam descobertos de outra forma. Em particular, projeções possibilitam que visualizemos como cada observação se relaciona às demais em um conjunto de dados, o que pode ser útil para identificar se e como diferentes observações se agrupam, quão próximos os grupos estão uns dos outros, se estes grupos são mais ou menos dispersos, e assim por diante. Durante mais de cem anos muitas técnicas foram desenvolvidas, e ainda há técnicas sendo desenvolvidas atualmente. No entanto, o problema de redução de dimensionalidade está longe de ser resolvido – não há uma técnica que possa ser considerada 'perfeita' em todas as situações. Além disso, o problema de como avaliar se uma projeção é boa ou não também está em aberto, já que não existe uma única métrica que seja capaz de apontar claramente o que é uma boa projeção.

Nesta tese, nós estendemos o estado da arte na área de redução de dimensionalidade em três direções principais, a saber.

Em primeiro lugar, nós fornecemos um *benchmark* quantitativo para verificar a qualidade das técnicas de projeção. Nosso *benchmark* avalia uma quantidade maior de técnicas, bases de dados, conjuntos de parâmetros e métricas do que qualquer outro *benchmark* conhecido por nós neste momento. Isto nos dá uma visão clara da área, e traz *insights* importantes sobre o quão boas são as técnicas e métricas existentes. Nosso *benchmark* também oferece uma metodologia concreta e detalhada, que pode ser utilizada por usuários que desejam escolher técnicas ótimas para os seus próprios problemas, de modo a satisfazer seus requisitos práticos.

Em segundo lugar, nós revisitamos uma área de pesquisa que não se desenvolveu muito desde o surgimento dos Mapas Auto-Organizáveis (*Self-Organizing Maps*) nos anos 1980: o uso de redes neurais aplicadas a projeções multidimensionais. Nós mostramos como redes neurais podem ser utilizadas para criarmos projeções que são melhores do que as existentes em termos de escalabilidade, capacidade de inferência e de mapeamento inverso, separação de grupos, e ainda, que sejam simples de serem usadas.

Por último, nós apresentamos várias aplicações de projeções multidimensionais que mostram sua importância como ferramentas para análise de dados multivariados, com foco na compreensão e melhoria do comportamento de classificadores de *Machine Learning* e de *solvers* para problemas de otimização.

PUBLICATIONS

This thesis is the result of the following publications:

- M. Espadoto, R. Martins, A. Kerren, N. S. T. Hirata, A. C. Telea. *Towards a quantitative survey of dimension reduction techniques* [56]
- M. Espadoto, E. Vernier, A. C. Telea. *Selecting and Sharing Multidimensional Projection Algorithms: A Practical View* [60]
- M. Espadoto, N. S. T. Hirata, A. C. Telea. *Deep Learning Multidimensional Projections* [52]
- M. Espadoto, F. C. M. Rodrigues, N. S. T. Hirata, R. Hirata Jr, A. C. Telea. *Deep Learning Inverse Multidimensional Projections* [59]
- M. Espadoto, F. C. M. Rodrigues, A. C. Telea. *Visual Analytics of Multidimensional Projections for Constructing Classifier Decision Boundary Maps*[1] [58]
- F. C. M. Rodrigues, M. Espadoto, R. Hirata Jr, A. C. Telea. *Constructing and Visualizing High-Quality Classifier Decision Boundary Maps* [170]
- M. Espadoto, A. Falcão, N. S. T. Hirata, A. C. Telea. *Improving Neural Network-based Multidimensional Projections* [51]
- M. Espadoto, N. S. T. Hirata, A. C. Telea. *Self-Supervised Dimensionality Reduction with Neural Networks and Pseudo-labeling* [53]
- M. Espadoto, F. C. M. Rodrigues, N. S. T. Hirata, A. C. Telea. *OptMap: Using Dense Maps for Visualizing Multidimensional Optimization Problems*[2] [54]

Other articles developed but still under revision at the moment of the writing include:

- T. S. Modrakowski, M. Espadoto, A. Falcão, N. S. T. Hirata, A. C. Telea. *Improving Deep Learning Projections by Neighborhood Analysis*
- Z. Tian, X. Zhai, D. van Driel, M. Espadoto, A. C. Telea. *Using Multiple Attribute-Based Explanations of Multidimensional Projections to Explore High-Dimensional Data*

1 Best Paper Award at IVAPP 2019
2 Best Student Paper Award at IVAPP 2021

CONTENTS

1 INTRODUCTION

Data Visualization is the field of Computer Science concerned with the visual presentation of data with the use of computational techniques. As a field, it can be divided into the two main subfields of Scientific Visualization (SciVis) and Information Visualization (InfoVis). Scientific Visualization can be defined as "*the use of computers or techniques for comprehending data or to extract knowledge from the results of simulations, computations or measurements*", according to [137]. It can be seen also as a set of techniques used to present data commonly related to physical phenomena or simulations thereof in areas such as Engineering, Physics, Geosciences, Medicine, among others. This type of data typically has a spatial location, *i.e.*, represents measurements associated with some positions in space. As such, this data is most naturally visualized in its definition space, which is typically 2D or 3D. Information Visualization is the field concerned with the visual presentation of abstract data in a way that allows for it to be better understood. There are different but complementary definitions by authors in the literature, such as "*the depiction of information using spatial of graphical representations, to facilitate comparison, pattern recognition, change detection, and other cognitive skills by making use of the visual system*", by [84], or "*the representation and presentation of data that makes use of our visual perception abilities in order to amplify cognition*", by [103]. By abstract data, we mean here all data that does not have a direct spatial association, in contrast with the data typically used in Scientific Visualization. InfoVis data thus ranges broader than SciVis data, including spreadsheets, tables, relational databases, and graphs.

Among the many existing challenges in InfoVis, an important one is the visualization of *multidimensional* or *high-dimensional* data. Simply put, these are datasets where every observation consists of many measurements, the latter being also called variables, attributes, or dimensions. The qualificative 'many' is here important: When such a dataset has two to four such dimensions, one can readily use SciVis methods to visualize it; the fourth dimension, if present, can be mapped to time by *e.g.* animation techniques. When the dimension count is slightly larger, *e.g.* under roughly 10, one can still use SciVis methods, by generating a set of so-called 'small multiples', *i.e.*, several identically-designed visualizations which show, in turn, subsets of the available dimensions. However, when the dimension count exceeds 10, such approaches do not scale. This causes serious challenges for datasets that have hundreds up to thousands of dimensions, such as frequent nowadays in data science and machine learning applications. The InfoVis community has proposed several techniques to address the challenge of high-dimensional data visualization, such as glyphs [226], parallel coordinate plots [94], table lenses [164, 202], scatterplot matrices [16], dimensionality reduction techniques [56, 148], and multiple views linking the above visualization types [28].

1.1 MULTIDIMENSIONAL PROJECTIONS

In this thesis, we focus on dimensionality reduction (DR) techniques, also called *multidimensional projections*. We prefer, and shall use, the terms *multidimensional projection* or simply *projection* because to us they convey a more generic meaning of taking data from one space to another, regardless of the number of dimensions in each space, which cannot be said of the term dimensionality *reduction*, which implies a direction, from

larger to smaller dimensionality. Compared to the other high-dimensional visualization techniques, projections have several important advantages:

- They *scale* far better than all other comparable visualization techniques in terms of the visual space needed to depict both *dimensions* and *observations*. Indeed, projections do not require additional visual space for additional dimensions, which makes them directly usable for datasets with thousands, and in theory, an unlimited number of, dimensions. Separately, every observation is encoded by a single point in a projection. This is the minimal amount of screen space one can use to encode an observation. Both these properties of projections make them very suitable for visualizing 'big data' collections of millions of observations and hundreds of dimensions or more on a single computer screen;

- They are *simple* to use: Whereas other high-dimensional visualization techniques require the user to perform various interactions to *e.g.* sort or group dimensions or (re)arrange separate views in a small multiple set-up, projections fundamentally only need an input dataset to create their visual output. Interaction can still bring added value, but is not a prerequisite of the method. This simplifies their use, thereby making them attractive for wide audiences;

- They require only *basic* graphics infrastructure: In the limit, displaying a projection requires only the means to show a 2D (color-coded) scatterplot. This works well on virtually and graphical output display in existence, in contrast to more sophisticated techniques which require rich 3D rendering, navigation, and interaction;

Projections are known since many decades, starting with the introduction and use in computer science and data visualization of Principal Component Analysis (PCA) [97]. Since then, tens if not hundreds of different projection techniques have been proposed. These vary enormously, in many aspects, in their quest to render the high-dimensional data in the 'best' way in a lower number of dimensions.

However, what means to render such data in the 'best' way? What is a 'good' projection? The answer to this question is elusive. A good projection can be defined in terms of being able to preserve a certain aspect of the high-dimensional data it projects (examples hereof are given in Chapter 2); or in terms of being robust to small-scale perturbations in the input data – or, by extension hereof, being able to project very different data in the same way the original data was projected, thereby presenting the user a stable, trusted, easy to interpret, way to understand the high-dimensional space [20, 124]; or, being computationally scalable to (very) large datasets of many observations and/or dimensions; or, being very simple to use in practice, having a few (ideally, no) free parameters to tune; or, being readily available in open-source implementations in multiple programming languages. All above requirements are important. However, no *single* projection method satisfies them all, to our knowledge.

The proliferation of projection methods is, in our view, a signal to the fact that the 'stakeholders' involved therein could not, or did not, converge to a clear set of requirements of what a good projection method should entail. Part of this is *inherent*: Reducing a (very) high-dimensional space to a low (typically, 2D or 3D, for the sake of direct visualization) space is not trivial and, as many examples showed [10, 11, 22, 200], will create problems and interpretation errors for the user who views the low-dimensional scatterplot aiming to gain insights in the high-dimensional data. Whereas such errors have been studied [133], it is far from clear which errors are most critical for a certain *task* regarding the data analysis. Part of the problem is a matter of *history*: Projections

have been developed at the crossroads of machine learning, data science, and information visualization. These fields have not always identical aims, so the promoted methods could not satisfy the needs of users across all these fields. Finally, part of the problem is due to the *size* of the problem: Designing an 'universal' projection algorithm (or family thereof) would require, or have required, investments and efforts which are far beyond what typical researcher groups in the involved areas could afford.

Still, the projection landscape has evolved impressively in the last decade. Two salient highlights thereof are listed next (for an exhaustive list, we refer to Chapter 3).

The creation of t-SNE by [127] changed the landscape of DR techniques for visualization, by providing unmatched visual cluster segregation – a requirement that most, if not all, users, were asking for. Despite its wide adoption, t-SNE has several drawbacks: It is slow for larger datasets; its hyperparameters are notoriously difficult to properly tune to achieve good visualizations; and it does not have out-of-sample capability – meaning, it cannot project new data 'along' old, existing, data. Over the years many improvements were proposed, such as the addition of out-of-sample support [124], which unfortunately does not have currently a practical implementation; acceleration by using the Barnes-Hut algorithm [126], which was very successful and is used by most current implementations; and the appearance of t-SNE variants, such as Dynamic t-SNE [166], designed for time-dependent data; and Hierarchical SNE [158] and Approximated and User and Steerable SNE (At-SNE) [159], which are variants that allow users to pick how much detail they want, and therefore, establish a trade-off between accuracy and speed. Still, all these variants of t-SNE do not *fully* comply with the user requirements we listed earlier above.

Even more recently, the Uniform Manifold Approximation and Projection technique [138] used a very different theoretical approach than t-SNE, which in practice produces results comparable in quality of cluster segregation to those of t-SNE, with the added benefits of being much faster and having out-of-sample capability. However, UMAP suffers from the difficult of tuning hyperparameters.

1.2 OBJECTIVES

In the above, we outlined that projections are an established tool, of high value, for visualizing high-dimensional data. However, they have several limitations concerning aspects such as quality (evaluation), scalability, robustness/stability, ease of use, and ease of usage.

The main objective of this thesis is to address as many of the above-mentioned limitations as we can within our effort scope. We start with the simplest, and interestingly enough not yet addressed question: What is the state of the art of projections today? Putting it even more sharply: Which projection methods should a practitioner consider to use in his or her field, given the specific constraints of that field? With this in mind, we outline our first research question:

RQ1: *How do current projection methods fare concerning quality, stability, and ease of use? What can we teach the practitioner concerning the plethora of such methods available nowadays?*

As the reader possibly guesses, the answer to the above question will not single out any projection method that outdoes its counterparts in all measured aspects. As such, the question remains open: Can we do better, concerning projections, than existing state-of-

the-art, and if so, how? More precisely, we single out (again) the following aspects in which we want to improve projections:

- *Scalability:* Few, or no, methods are capable of projecting large data (*e.g.*, $\geq 1M$ observations, ≥ 100 dimensions) time (*e.g.*, seconds, something needed for interactive data exploration;
- *Out-of-sample capability:* Most existing projection methods are not parametric, *i.e.*, they cannot project unseen data without re-running the algorithm from scratch; simply put, when new data comes in, these methods create an *entirely* different projection as compared to the one of the original data. As such, users cannot 'place in' new samples in the new projection around old samples. Their mental map is lost, meaning, the added-value of the projection method is low;
- *Inverse projection capability:* Most projection methods cannot take points in 2-dimensional space and project them into the original high-dimensional space. This severely limits the use of projections to 'only' show an existing dataset. Reasoning *outside* this dataset is not possible. One cannot know what is 'between' the 2D scatterplot points shown by a projection.

Summarizing the above, we outline our second research question:

RQ2: *How can we create projection algorithms which can handle millions of observations with hundreds of dimensions at interactive rates; project new, unseen data, in the same way old data was projected; and tell users what data points belong to the empty spaces between a projection's scatterplot points?*

As, again, the reader guesses, no existing projection method can do the above. Hence, the challenge of creating such a method is a valid, and viable, one.

1.3 CONTRIBUTIONS

In this thesis, we approach the research questions RQ1 and RQ2 outlined above, with the following contributions:

- A comprehensive, quantitative evaluation and survey of existing multidimensional projection methods (answering RQ1);
- A method that enables the user to mimic existing projection methods with desirable visual characteristics by using supervised learning, adding out-of-sample and inverse projection capabilities and making it scalable to large datasets (first answer to RQ2);
- A method that has the aforementioned capabilities, which can be trained by using unsupervised learning (refined answer to RQ2);
- Several applications where multidimensional projections can help the process of data analysis and pattern recognition (practical answer to RQ1 and RQ2).

1.4 ORGANIZATION

This thesis is organized into several chapters, which address the above research questions as follows:

Chapter 2 presents preliminary information that will be used across the entire thesis, such as notation and quality metrics for projections, among others.

Chapter 3 presents a quantitative evaluation and survey of the state-of-the-art in terms of dimensionality reduction techniques, thereby targeting RQ1. Chapter 4 presents the framework designed in order to obtain the results shown in Chapter 3, thereby providing hands-on information on how we managed to complete the daunting evaluation of tens of projection methods, datasets, and quality metrics, from a practical perspective. It also serves to help interested readers that want to extend our evaluation work and/or design alternative large-scale evaluations of projections.

Chapter 5 presents Neural Network Projection (NNP), a technique that uses neural networks to learn how to mimic different DR techniques, which provides, among other things, the ability of doing fast inference for non-parametric techniques. This is our first answer to RQ2. Chapter 6 presents k-Nearest Neighbor Network Projection (kNNP), a neighborhood-based version of the NNP. This refines the proposal in Chapter 5 showing increased quality metric values for the projection.

Chapter 7 presents an inverse projection technique based on neural networks called NNInv. This completes our answer to RQ2 by making the mapping from the projection space to the data space practical and computationally efficient.

Chapters 8 and 9 present applications that use our proposed direct and inverse projection techniques for a variety of use cases. First, we show how we can use our proposals to visualize decision maps of machine learning classifiers (Ch. 8). Next, we show how those techniques can be used to visually explore multivariate optimization problems (Ch. 9).

Chapter 10 puts together ideas from previous chapters and refines the strive to creating good-quality projections by presenting Self-Supervised Network Projection (SSNP), an approach based on autoencoders and clustering algorithms that improves the results generated by standard autoencoders, while being able to produce direct and inverse projections and to mimic the base clustering algorithm as well, all done by training the algorithm only once.

Finally, Chapter 11 concludes the thesis, by outlining our conclusions and directions for future work.

2 PRELIMINARIES

In this chapter we present a detailed reference of notations, quality metrics and datasets mentioned in this thesis.

2.1 NOTATION

Let $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i \in \mathbb{R}, 1 \leq i \leq n$ be a n-dimensional (nD) real-valued observation or sample, and let $D = \{\mathbf{x}_i\}$, $1 \leq i \leq N$ be a nD dataset of N observations. Let $\mathbf{x}^j = (x_1^j, \ldots, x_N^j)$, $1 \leq j \leq n$ be the j^{th} feature vector of D. Thus, D can be seen as a table with N rows (observations) and n columns (features or dimensions). A projection technique, or algorithm, is then a function

$$P : \mathbb{R}^n \rightarrow \mathbb{R}^q, \tag{2.1}$$

where $q \ll n$, and typically $q = 2$. P can also have p so-called free parameters, or hyperparameters, π_i, $1 \leq i \leq p$, which can be tuned by the end user to obtain different trade-offs of P. The projection $P(\mathbf{x})$ of a sample $\mathbf{x} \in D$ is a qD point. Projecting an entire dataset D yields a qD scatterplot, denoted next for brevity as $P(D)$. We denote by cursive letters the power set (set of all sets) of a given type, *e.g.*, $\mathcal{D}$ is the set of all nD datasets D, and $\mathcal{P}$ is the set of all projection techniques P. The inverse of P, denoted $P^{-1}(\mathbf{p})$, maps a qD point $\mathbf{p}$ to the high-dimensional space $\mathbb{R}^n$. Note that we do not make – unless explicitly said – any assumption about what $\mathbf{p}$ is; it can be a point resulting from projecting some sample in a dataset D, or any other point in $\mathbb{R}^q$.

2.2 QUALITY METRICS

To capture the quality of a projection technique P, let

$$M : \{(D \in \mathcal{D}, P(D))\} \rightarrow \mathbb{R}^k \tag{2.2}$$

be a metric that assigns to the pair formed by dataset D and its projection $P(D)$ a scalar ($k = 1$) or vector ($k > 1$) value. Let $\mathcal{M}$ be the set of all such metrics. Different metrics M capture different desirable aspects of a projection P. The key one, that all techniques consider, is preserving *similarity* of points when projecting from nD to qD. This is usually defined as Euclidean, geodesic, Procrustes, or cosine distance, or the probabilities of a point to have the same neighbors in $\mathbb{R}^n$ and $\mathbb{R}^q$ [29, 127]. Besides similarity, other quality aspects include computational scalability, ease of use (*vs* parameter setting), and robustness *vs* small input-data changes or hyperparameter changes. See Section 3.4 for a further discussion about quality metrics.

Formally, to compare several projection techniques, we need to understand the distribution of *all* values of $\mathcal{M}$ over *all* values of $\mathcal{D}$ and $\mathcal{P}$, *i.e.*, how all quality metrics M vary over all combinations of datasets and projection techniques. Fully computing this distribution is practically impossible, since the spaces $\mathcal{P}$ and $\mathcal{M}$ have a very high cardinality, while $\mathcal{D}$ is infinite. All projection evaluation papers handle this by *sampling* $\mathcal{D}$, and $\mathcal{P}$, and $\mathcal{M}$ to select a small subset of datasets $\overline{D} \subset \mathcal{D}$, techniques $\overline{P} \subset \mathcal{P}$, and metrics

$\overline{M} \subset \mathcal{M}$ to evaluate over. We call such a subset $B = \overline{D} \times \overline{P} \times \overline{M}$ a *benchmark*. An *evaluation* of a benchmark is thus the multidimensional set of values

$$E = \{M(D, P(D)) | (D, P, M) \in B\}. \tag{2.3}$$

To evaluate the quality of the obtained projections, we used the metrics in Table 1, which are well-known in the DR literature. As per Equation (2.2), we classify metrics based on their output dimensionality (k value):

Table 1: Quality metrics. In the Range column, optimal value is marked in bold; for unbounded metrics smaller values are better.

Metric	Definition	Range	Type
Trustworthiness (M_t)	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in U_i^{(K)}} (r(i,j) - K)$	$[0, \mathbf{1}]$	scalar
Continuity (M_c)	$1 - \frac{2}{NK(2n-3K-1)} \sum_{i=1}^{N} \sum_{j \in V_i^{(K)}} (\hat{r}(i,j) - K)$	$[0, \mathbf{1}]$	scalar
Neighborhood hit (M_{nh})	$\frac{1}{N} \sum_{\mathbf{y} \in P(D)} \frac{\mathbf{y}_k}{\mathbf{y}_k}$	$[0, \mathbf{1}]$	scalar
Shepard diagram (S)	Scatterplot $(\|\mathbf{x}_i - \mathbf{x}_j\|, \|P(\mathbf{x}_i) - P(\mathbf{x}_j)\|), 1 \le i \le N, i \neq j$	-	point-pair
Shepard diagram correlation (M_S)	Spearman rank correlation of Shepard diagram	$[0, \mathbf{1}]$	scalar
Normalized Stress (M_σ)	$\frac{\sum_{ij}(\mathbf{D}_{ij} - \mathbf{d}_{ij})}{\sum_{ij} \mathbf{D}_{ij}^2}$	$[\mathbf{0}, 1]$	scalar
Average local error ($M_a(i)$)	$\frac{1}{N-1} \sum_{j \neq i} \left\| \frac{\Delta^n(\mathbf{x}_i, \mathbf{x}_j)}{\max_{i,j} \Delta^n(\mathbf{x}_i, \mathbf{x}_j)} - \frac{\Delta^q(P(\mathbf{x}_i), P(\mathbf{x}_j))}{\max_{i,j} \Delta^q(P(\mathbf{x}_i), P(\mathbf{x}_j))} \right\|$	$[\mathbf{0}, 1]$	local (per-point)
Mean squared error	$\frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2$	Unbounded	scalar
Kullback-Leibler Divergence	$\sum \mathcal{P}(D) log\left(\frac{\mathcal{P}(D)}{\mathcal{Q}(P(D))}\right)$, where $\mathcal{P}$ and $\mathcal{Q}$ denote probability distributions	Unbounded	scalar
Cross-entropy	$\sum \mathcal{P}(D) log \mathcal{Q}(P(D))$, where $\mathcal{P}$ and $\mathcal{Q}$ denote probability distributions	Unbounded	scalar
LLE Reconstruction Error	$\sum_i \|P(D)_i - \sum_j W_{ij} P(D)_j\|^2$	Unbounded	scalar
Isomap Cost Function	$\frac{\|K(\mathbf{D}) - K(\mathbf{d})\|}{n}$, where $K(D) = -0.5(I - \frac{1}{n}) D^2 (I - \frac{1}{n})$ and n is the number of observations	Unbounded	scalar

2.2.1 *Scalar metrics*

The simplest and most used quality metrics yield a scalar value ($k = 1$, Equation (2.2)) for a projection $P(D)$. We choose the following scalar metrics, given that they are well known, easily interpretable, and used in most DR papers. Table 1 lists their definitions.

Trustworthiness (M_t) [208]: Measures the fraction of points in D that are also close in $P(D)$. M_t tells how much one can trust that local patterns in a projection, *e.g.* clusters, represent actual patterns in the data. In the definition (Table 1), $U_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in the 2D space but not among the K nearest neighbors of point i in $\mathbb{R}^n$; and $r(i, j)$ is the rank of the 2D point j in the ordered-set of nearest neighbors of i in 2D. We choose $K = 7$, in line with [128, 134];

Continuity (M_c) [208]: Measures the fraction of close points in $P(D)$ that are also close in D. In the definition (Table 1), $V_i^{(K)}$ is the set of points that are among the K nearest neighbors of point i in $\mathbb{R}^n$ but not among the K nearest neighbors in 2D; and $\hat{r}(i, j)$ is the rank of the $\mathbb{R}^n$ point j in the ordered set of nearest neighbors of i in $\mathbb{R}^n$. As with

M_t, we choose $K = 7$;

Neighborhood Hit (M_{nh}) [154]: Measures how well-separable labeled data is in a projection $P(D)$, in a rotation-invariant fashion, from perfect separation ($M_{nh} = 1$) to no separation ($M_{nh} = 0$). M_{nh} is defined as the number $\mathbf{y}_k^l$ of the k nearest neighbors of a point $\mathbf{y} \in P(D)$, denoted by $\mathbf{y}_k$, that have the same label as $\mathbf{y}$, averaged over $P(D)$. In this thesis we used $k = 7$, unless otherwise noted;

Shepard diagram correlation (M_s) [95]: The Shepard diagram is a scatterplot of the pairwise (Euclidean) distances between all points in $P(D)$ *vs* the corresponding distances in D. The closer the plot is to the main diagonal, the better overall distance preservation is. Plot areas below, respectively above, the diagonal indicate distance *ranges* for which false neighbors, respectively missing neighbors, occur. We quantitatively assess a Shepard diagram by computing its Spearman rank correlation M_s. A value of $M_s = 1$ indicates a perfect (positive) correlation of distances;

Normalized stress (M_σ) [95]: Measures the difference between the distance matrices of points in D and in $P(D)$ respectively;

Mean squared error: Measures the mean squared error between D and the inverse of the transformation computed by an inverse projection technique, *i.e.* $P^{-1}(P(D))$;

Kullback-Leibler divergence [127]: Measures the difference between two probability distributions created from D and $P(D)$, which can be viewed as the relative entropy between two distributions;

Cross entropy [138]: Similar to the Kullback-Leibler divergence, cross entropy also measures the difference between two probability distributions created from D and $P(D)$, but as the total entropy between two distributions;

LLE Reconstruction Error [175]: Measures the reconstruction error of $P(D)$ based on LLE (Locally Linear Embedding) computed weights and cost function;

Isomap cost function [203]: Measures the Isomap cost function based on the distance matrices $\mathbf{D}$ and $\mathbf{d}$ of points in D and in $P(D)$ respectively.

The last five metrics (mean squared error, Kullback-Leibler divergence, cross entropy, LLE reconstruction error and Isopam cost function) are used in this thesis only in specific cases, and are referred to by S_t where applicable.

2.2.2 *Point-pair metrics*

While simple to compute and interpret, scalar metrics average a projection's quality over all its points. Comparing DR techniques using only averages is either misleading or not insightful enough. This was recognized by Joia *et al.* [95] when comparing the distortions of (Euclidean) distances caused by several projection techniques, and further elaborated by Nonato *et al.* [148]. For example, two projections may have similar average distortions, but one may preserve small distances better than the other, which makes it more suitable for, *e.g.*, cluster analysis. To capture such aspects, point-pair metrics measure properties of every point pair in the data (and projection result), as

follows.

Shepard diagram (S) [95]: The Shepard diagram is a scatterplot of the pairwise (Euclidean) distances between all points in $P(D)$ *vs* the corresponding distances in D. The closer the plot is to the main diagonal, the better overall distance preservation is. Plot areas below, respectively above, the diagonal indicate distance *ranges* for which false neighbors, respectively missing neighbors, occur. We quantitatively assess the quality of a Shepard diagram by computing its Spearman rank correlation M_s. A value of $M_s = 1$ indicates a perfect (positive) correlation of distances.

Other point-pair metrics include the co-ranking matrix [114] of the pairwise (Euclidean) distances between all points in $P(D)$ *vs* corresponding distances in D. It is related to the Shepard diagram as both their main diagonals can be interpreted similarly. The co-ranking matrix allows analyzing false and missing neighbors. Yet, summarizing this matrix to a value that is simple to interpret, as we did for the Shepard diagram using the Spearman rank, is harder. Hence, we did not include this metric in our study.

2.2.3 *Local Metrics*

Both scalar and point-pair metrics are *sample-agnostic*, *i.e.*, they do not tell how projection errors correlate with specific samples or sample groups. Knowing this is important to assess which *patterns* in a projection $P(D)$ one can trust and which not. Several so-called spatial distribution, visual, distortion, or local, metrics have been proposed for this. These take different values for each point in $P(D)$, *i.e.* have $k = N$ in Equation (2.2), as follows.

Projection precision score [179]: This is the normalized distance between the two k-dimensional vectors having as components Euclidean distances between a point $\mathbf{y} \in P(D)$ and its K nearest neighbors in D, respectively $P(D)$, visualized by color-coding $P(D)$. Yet, this metric cannot differentiate false from missing neighbors;

Stretching and compression [9, 115]: These measure the increase (stretching), respectively decrease (compression) of distances of a point $\mathbf{y} \in P(D)$ *vs* all other points in $P(D)$ *vs* the corresponding distances in D. These metrics are visualized using a Voronoi-based partitioning of the 2D projection space which, as the authors note, may lead to bias due to how Voronoi cells depend on small perturbations of their underlying sites;

Average local error (M_a) [133]: This metric assigns, for each point i, the averaged sum $M_a(i)$ of differences between its normalized distances in $\mathbb{R}^n$ and $\mathbb{R}^q$ to all other points j in the dataset (Table 1). $M_a(i)$ ranges in $[0, 1]$. Small values indicate good placement of point i *vs* all other points. This metric has been adapted to also show neighborhood preservation [134]. It is typically displayed using heat maps.

Local metrics show subtle differences between DR techniques. Yet, they need more presentation space in contrast to scalar and point-pair metrics, and also need to be visually (manually) assessed.

2.3 DATASETS

For every dataset used in this thesis, we present here a short description:

Bank Marketing [143]: 45211 observations described with 17 attributes, extracted from a direct marketing campaign of a Portuguese bank used to predict whether a client will subscribe to a banking product or not;

CIFAR-10 and CIFAR-100 [105]: 60000 observations of animals and vehicles rendered as 32x32-pixel color images and divided into 10 and 100 classes, respectively. We used the DenseNet [91] CNN pre-trained on the ImageNet dataset to extract features of those images, yielding 1920-element vectors for each image;

CNAE-9 [36]: 1080 observations of free text descriptions of Brazilian companies in the National Classification of Economic Activities, split in 9 classes based on economic activity and described with 857 attributes;

COIL20 [146]: Columbia University Image Library, consisting of 1440 images of 20 types of common objects, described with 400 attributes;

Dogs vs Cats [49]: 25000 images of varying sizes divided into two classes (cats, dogs). We used the Inception V3 [196] Convolutional Neural Network (CNN) pre-trained on the ImageNet data set [45] to extract features of those images, yielding 2048-element vectors for each image;

Epileptic Seizure Recognition [7]: 11500 observations from brain activity used to detect epileptic seizures, described with 178 attributes;

Fashion MNIST [223]: 70000 images of 10 types of pieces of clothing, rendered as 28x28-pixel gray scale images, flattened to 784-element vectors;

Flickr Material Database [186]: 1000 images of common materials used for training material recognition systems. We used the Inception Resnet V2 [35] Convolutional Neural Network (CNN) pre-trained on the ImageNet data set [45] to extract features of those images, yielding 1536-element vectors for each image;

HIVA [78] : 21339 observations divided into three classes, used to predict which chemical compounds are active against HIV infection, described with 1617 attributes;

Hate Speech [44]: 24802 tweets labeled according to the type of offensive language they contain, used for training hate speech detectors, described with 100 attributes;

Human Activity Recognition [8]: 10299 observations from 30 subjects performing activities of daily living used for human activity recognition, described with 561 dimensions;

IMDB Movie Review [122]: 25000 movie reviews from which 700 attributes were extracted using TF-IDF [176], a standard method in text processing;

MNIST [112]: 70000 images of handwritten digits from 0 to 9, rendered as 28x28-pixel gray scale images, flattened to 784-element vectors;

ORL [177]: 400 gray scale face images from 40 different subjects, described with 396 attributes;

Reuters Newswire Dataset [204]: 8432 observations of news report documents, from which 5000 attributes were extracted using TF-IDF [176], a standard method in text processing. This is a subset of the full dataset which contains data for the six most frequent classes only;

SECOM [136]: 1567 observation from a semiconductor manufacturing process described with 590 attributes, used for training failure detectors;

SMS Spam Collection [4]: 5574 observations from SMS labeled messages collected for mobile phone spam research, used for training SMS spam detectors, described with 500 attributes;

Seismic Bumps [187]: 2584 observations, used to forecast seismic bumps in a coal mine, described with 24 attributes;

Sentiment Labeled Sentences [104]: 3000 sentences, used for sentiment analysis, described with 200 attributes;

Spambase [90]: 4601 observations of email classified as spam or not spam, described with 57 attributes;

Street View House Numbers [147]: 73257 images of digits 0 to 9 extracted from Google Street View, rendered as 32x32 pixel color images, flattened to 3072-element vectors;

Wisconsin Breast Cancer - Diagnostic [149]: 569 observations of a fine needle aspirate (FNA) of a breast mass, described with 32 attributes;

Apart from the above, in some experiments we use synthetic datasets, which are detailed in the context where they appear.

3 SURVEY OF EXISTING TECHNIQUES

Exploring high-dimensional data is central to many application domains such as statistics, data science, machine learning (ML), and information visualization (InfoVis). The main difficulty encountered in this task is the large size of such datasets, both in the number of observations (also called samples) and measurements recorded per observation (also called dimensions, features, variables, or attributes). As such, high-dimensional visualization has become an important sub-field of Information Visualization [89, 100, 119, 198].

Several techniques exist for high-dimensional data visualization, including glyphs [226], parallel coordinate plots [94], table lenses [164, 202], scatterplot matrices [16], dimensionality reduction methods [148], and multiple views linking the above visualization types [28]. In this family, *dimensionality reduction* (DR) methods, also called projections, have a particular place: compared to other techniques, they scale much better in terms of both the number of samples and the number of dimensions they can show on a given screen space area. As such, projections have become the tool of choice for exploring data which has a high number of dimensions (tens up to hundreds) and/or in applications where the individual identity of dimensions is less important, as in *e.g.* machine learning applications. In the last decade, many projection techniques have been proposed [128, 148, 192], of which t-SNE [127] is arguably one of the best known and most adopted by applications.

This explosion of the number and variety of projection techniques and their widespread use in many applications makes it hard for end users to understand how to *choose* a good technique for a given use context. Several functional and non-functional requirements must be considered, such as the ability of the projection to preserve certain patterns (*e.g.*, neighbors, distances, or clusters); doing this for a given number of dimensions (which can be low or very high); computational scalability in both observation and dimension counts; robustness to small changes in both data and algorithm parameters, *i.e.*, yielding similar results for small changes of these inputs; ease of use in terms of number and complexity of settings asked from the end user; and available implementations. Current literature addresses such questions by comparative studies (*e.g.*, in papers that propose new projection techniques), best-practice studies, or survey papers. Yet, such approaches have limitations: Technique papers typically cover only a few techniques; survey papers consider tens of techniques, but typically focus on high-level and/or more theoretical aspects, and less on benchmarking many projection techniques on combinations of datasets, technique parameter settings, and evaluating quality metrics. Best-practice studies fall somewhere in the middle.

This chapter presents a survey which aims to address the above-mentioned limitations, as follows (see also Fig. 3.1). First, we overview related surveys in evaluation and comparison of DR methods (Section 3.1). Based on these, we propose taxonomies covering the types of multidimensional datasets, projection techniques, and quality metrics used to assess these. This way, we explicitly show which parts of the data, projection, and quality spaces we next cover, and how. We model these taxonomies based on a number of so-called *traits* of datasets, techniques, and metrics respectively (we use the term traits to avoid confusion with dimensions). Next, we sample these spaces by 18 datasets, 44

This chapter is based on publication [56].

techniques, and 7 quality metrics respectively, to create a projection assessment benchmark. The respective taxonomies, their traits, and how these are sampled to yield our benchmark are discussed in Sections 3.2, 3.3, and 3.4 respectively. We run this benchmark, using an optimization strategy to find the best projection-technique parameter values for the considered quality metrics (Section 3.5). Finally, we present and discuss the obtained measurements (Section 3.6). We outline several observations on the correlations between dataset types, projection techniques, and quality aspects. Next, we select a few special cases (points of interest in our data, projections, and quality space) and examine these in more detail (Section 3.6). Section 3.7 discusses the main findings and limitations of our survey. We conclude the chapter by outlining directions of future work (Section 3.8).

We next discuss how existing surveys design evaluations E, *i.e.*, which decisions they take to sample the continuous spaces $\mathcal{D}$, $\mathcal{P}$, and $\mathcal{M}$ to evaluate B. We next propose ways to extend this state of the art in Sections 3.2-3.4.

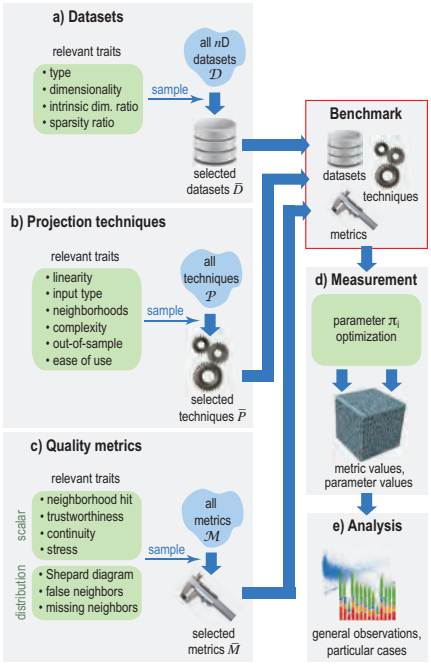


Figure 3.1: Workflow of survey outlining its main stages: Construction of benchmark of datasets (a), projection techniques (b), and quality metrics (c) based on taxonomies of these spaces; measurement of metric values using parameter optimization (d); and analysis of results (e).

3.1 RELATED WORK

In this section, we discuss related work on comparisons and evaluations of DR techniques. We do not detail here all projection techniques and quality metrics (and related papers) — this is done in context in Sections 3.3 and 3.4, respectively. Several surveys that compare DR techniques for visualization (and sometimes beyond) have been pub-

lished. We discuss these in chronological order. Since DR is at the crossroads of infovis and ML, we group surveys accordingly (See Table 2).

3.1.1 *Surveys from Machine Learning*

Projection techniques are known and used since decades in ML [97, 206]. Fodor [65] presents the earliest survey on projection techniques that we are aware of, which includes what are now considered modern methods, *i.e.*, nonlinear methods, vector quantization, and neural networks. This survey discusses 12 projection methods, including linear (PCA, ICA, FA) and some nonlinear (SOM, VQ, and NN) methods. While the survey (briefly) outlines the techniques underlying these methods, no actual side-by-side evaluation or quality metrics are discussed.

Yin [227] proposes a survey for nonlinear DR, focusing on visualization, covering seven projection methods. It also discusses variants of the stress metric for measuring projection quality. Yet, only two simple datasets are evaluated, using only four of the seven DR methods.

Maaten *et al.* [128] present, to our knowledge, the first systematic theoretical and *practical* comparison of PCA (linear) and other 13 (nonlinear) DR techniques. The theoretical side discusses the number of parameters, computational and memory complexities, and out-of-sample ability (whether a projection can handle new samples based on existing projected ones). Practical comparison includes measuring three scalar metrics (trustworthiness, continuity, and preservation of closest neighbors [208]) on 5 artificial and 5 real-world datasets. Optimal parameters were found using grid search. However, how the three quality metrics listed above were merged into a single quality (cost) function to optimize by grid search is not detailed. Also, the survey does not cover many well-known projection techniques (Table 2).

Bunte *et al.* [29] propose a theoretical framework to unify nine existing projection techniques. These are compared in terms of how *similarity* between points in nD and 2D is defined; which error metric the projection P minimizes; and which additional constraints the methods have. The techniques are evaluated on three datasets ranging between a few thousand and 20K points having between 16 and 36 dimensions. However, many existing projection techniques are not covered by this survey.

Sorzano *et al.* [192] present one of the most complete surveys from the viewpoint of number of discussed DR techniques – around 30, including variants of some main techniques. Yet, this survey has mainly a theoretical focus. Heuristics and cost functions underlying the DR techniques are discussed, but practical evaluation involves only a *single* image showing how LLE, HLLE, and ISO perform on a small synthetic dataset of 1K points in 3 dimensions. Measurements of quality metrics are not given.

Gisbrecht *et al.* [71] evaluate the suitability for data visualization of 10 DR techniques on 3 synthetic datasets of 1K three-dimensional points each. Projection quality is defined as a single scalar value using the rank-based criteria in [114]. Compared to earlier surveys, this one includes assessing computational scalability; and focuses on "popular" projection techniques, as these are more likely to be used in practice, so understanding how they perform is of increased added value. Yet, the evaluation confidence is limited by the very small number of tested datasets.

Cunningham *et al.* [40] present an excellent survey of *linear* DR techniques. This work is very similar in goals and structure to Sorzano *et al.* [192], *i.e.*, it aims to compare 15 DR techniques and a few sub-variants thereof from theoretical and mathematical viewpoints. No practical evaluations of quality metrics of existing techniques on datasets are given. Also, nonlinear projections are not considered.

Finally, Xie *et al.* [224] survey DR techniques based on the Random Projection (RP) method [43]. Such methods are arguably better at keeping data structure and/or reducing computational effort when dealing with a high dimension count. About 25 RP variants and a few sub-variants are discussed from a general perspective. This survey aims to provide a "reading map" for the RP literature. Yet, no side-by-side evaluation of existing methods on a benchmark, using specific quality metrics, is given.

3.1.2 *Surveys from InfoVis*

The infovis literature is rich in papers that evaluate projections. We next focus on key papers that share the aim of our work (comparing projections from a quantitative perspective). Additional papers related to assessing projection quality are discussed later in context.

Buja *et al.* [28] present one of the earliest surveys on projection usage to visualize multidimensional data. They propose a task-based taxonomy of interaction techniques for analyzing high-dimensional data. Projections, implemented in the XGobi tool [195], are just one of the considered techniques, in a linked-view set-up, to validate the proposed interaction taxonomy. In a related work, Hoffman *et al.* [89] compare 15 visualization techniques on two small datasets (hundreds of observations, 4 to 6 dimensions). Among these, three are projection techniques (Sammon's mapping (SAM), MDS, and Kohonen's self-organizing maps (SOM)). This survey does not contain any quantitative comparison of the discussed techniques.

Engel *et al.* [50] propose "an introduction to dimension reduction from a visualization point of view". They propose a taxonomy that compares nine DR methods from the viewpoint of their online behavior (out-of-sample ability) and computational complexity. Yet, no actual evaluation of quality metrics on datasets is given. The survey strongly makes the point that optimal parameter setting is an important but not well explored aspect that influences the quality, and finally usability, of projection techniques. We address this aspect in our work (Section 3.5).

Kehrer *et al.* [100] present a survey of methods for visualization of so-called "multi-faceted" scientific data. The survey overviews the context in which projection techniques are used in the (much) broader scope of visual analysis of multidimensional, multi-source, and multi-type datasets. Given this broad scope, details concerning the evaluation of projection techniques are not given. Liu *et al.* [119] present a related survey focused more specifically on visualizing high-dimensional data. They propose a 14-element taxonomy of techniques for high-dimensional visualization, of which dimensionality reduction is one. They also briefly introduce several projection quality metrics, such as global stress, local stress [185], ranking discrepancy [114, 121], and the projection precision score [179]. While seven concrete projection techniques are named, evaluating these and/or the aforementioned metrics is not covered.

Close to our goals, Nonato and Aupetit [148] survey the use of projections in visual analytics (VA) tasks. Their work, which is arguably together with [128] one of the most extensive and detailed surveys in DR literature, propose a taxonomy where 28 projection techniques are classified along their input data types, linearity, flexibility for supervision (label data), handling multilevel structures, locality, steerability, stability, and ability to handle out-of-code (large) data. They also discuss 14 projection quality metrics. Yet, actual *measurements* of how techniques perform, with respect to metrics, on a representative benchmark of datasets, is not in the scope of this survey. Our work aims to fill in this gap. On the other side, [148] covers several other directions, most notably the relation between DR techniques, caused distortions, VA tasks affected by distortions, and visual

enrichments that can alleviate such problems. All these aspects are not in the scope of our work.

Table 2: Projection techniques discussed in surveys on dimensionality reduction from machine learning and Infovis. The last column corresponds to this chapter, and the last row shows total number of techniques discussed in each survey. See Section 3.1.

Acronym	Projection Full Name	Fodor[65]	Hoffman[89]	Yin[227]	Maaten[128]	Bunte[29]	Engel[50]	Sorzano[192]	Cunningham[40]	Gisbrecht[71]	Liu[119]	Xie[224]	Nonato[148]	**Ours**
AE	Autoencoder				•									•
CCA	CCA (Canonical Correlations Analysis)								•					
CHL	Chalmers												•	
CLM	ClassiMap												•	
CuCA	CCA (Curvilinear Component Analysis)												•	
DM	Diffusion Maps				•									•
DML	Distance Metric Learning								•					
EM	Elastic Maps							•						
FA	Factor Analysis	•						•	•					•
FD	Force-Directed												•	
FMAP	FastMap												•	•
FS	Feature Selection											•		
GDA	Generalized Discriminant Analysis													•
GPLVM	Gaussian Process Latent Variable Model													•
GTM	Generative Topographic Mapping							•					•	
ICA	Independent Component Analysis	•						•	•					
F-ICA	FastICA													•
NL-ICA	Nonlinear ICA	•												
IDMAP	IDMAP													•
ISO	Isomap		•	•	•	•	•			•			•	•
L-ISO	Landmark Isomap													•
KECA	Kernel Entropy Component Analysis							•						
KLP	Kelp												•	
LAMP	LAMP												•	•
LDA	Linear Discriminant Analysis								•		•	•	•	
LE	Laplacian Eigenmaps				•	•				•	•		•	•
LLC	Locally Linear Coordination				•									•
LLE	Locally Linear Embedding		•	•	•	•	•			•	•		•	•
H-LLE	Hessian LLE				•									•
M-LLE	Modified LLE													•
LMNN	Large-Margin Nearest Neighbor Metric													•
LoCH	Local Convex Hull												•	
LPP	Locality Preserving Projection								•					•
LR	Linear Regression								•					
LSP	Least Square Projection												•	•
LTSA	Local Tangent Space Alignment				•								•	•
L-LTSA	Linear Local Tangent Space Alignment													•
MAF	Maximum Autocorrelation Factors								•					
MC	Manifold Charting				•					•				•
MCA	Multiple Correspondence Analysis												•	
MCML	Maximally Collapsing Metric Learning													•
MDS	Metric Multidimensional Scaling	•	•	•	•	•	•	•	•		•		•	•
L-MDS	Landmark MDS													•
MG-MDS	Multi-Grid MDS						•							
N-MDS	Nonmetric MDS (Kruskal)		•				•						•	•
ML	Manifold Learning						•							
MVU	Maximum Variance Unfolding				•	•				•			•	

Table 2 continued

Acronym	Projection Full Name	Fodor[65]	Hoffman[89]	Yin[227]	Maaten[128]	Bunte[29]	Engel[50]	Sorzano[192]	Cunningham[40]	Gisbrecht[71]	Liu[119]	Xie[224]	Nonato[148]		**Ours**
FMVU	Fast MVU														
L-MVU	Landmark MVU														•
NeRV	Neighborhood Retrieval Visualizer					•									
t-NeRV	t-NeRV					•									
NMF	Nonnegative Matrix Factorization							•	•						•
NLM	Nonlinear Mapping												•		
NN	Neural Networks	•													
PBC	Projection By Clustering														•
PC	Principal Curves	•						•							
PCA	Principal Component Analysis	•	•		•		•	•	•	•	•	•	•		•
I-PCA	Incremental PCA							•							•
K-PCA-P	Kernel PCA (Polynomial)														•
K-PCA-R	Kernel PCA (RBF)		•		•		•	•		•					•
K-PCA-S	Kernel PCA (Sigmoid)														•
L-PCA	Localized PCA							•							
NL-PCA	Nonlinear PCA	•		•				•							
P-PCA	Probabilistic PCA								•						•
R-PCA	Robust PCA							•							
S-PCA	Sparse PCA							•							•
PLMP	Part-Linear Multidimensional Projection												•		
PLP	Piecewise Laplacian-based Projection						•						•		
PLSP	Piecewise Least Square Projection														•
PM	Principal Manifolds			•											
PP	Projection Pursuit	•													
RBF-MP	RBF Multidimensional Projection												•		
RP	Random Projections	•										•			
G-RP	Gaussian Random Projection														•
S-RP	Sparse Random Projection														•
SAM	Sammon Mapping				•										
R-SAM	Rapid Sammon (Pekalska)												•		•
SDR	Sufficient Dimensionality Reduction								•						
SFA	Slow Feature Analysis								•						
SMA	Smacof												•		
SNE	Stochastic Neighborhood Embedding					•							•		
T-SNE	t-Dist. Stochastic Neighborhood Embedding					•				•	•				•
SOM	Self-Organizing Maps	•		•				•					•		
ViSOM	ViSOM (Visualization-induced SOM)			•											
SPE	Stochastic Proximity Embedding														•
G-SVD	Generalized SVD							•							
T-SVD	Truncated SVD														•
TF	Tensor Factorization							•							
UMAP	Uniform Manifold Approximation and Proj.														•
VQ	Vector Quantization	•						•							
Total		12	6	7	14	9	9	19	14	8	6	4	28		44

3.1.3 *Summary of Current Surveys*

Given related work in ML and infovis, we can state that current surveys do not cover several aspects of our goal. Table 3 overviews the number of evaluated DR techniques, number of datasets used for evaluation, and number of evaluated metrics. We see that

Table 3: Summary of surveys on dimensionality reduction from both machine learning (ML) and InfoVis (IV) listed in Table 2, with the respective number of DR techniques (discussed), datasets (used in evaluation), and metrics (computed on the datasets). The last column corresponds to this chapter. See Section 3.1.3.

	Fodor[65]	Hoffman[89]	Yin[227]	Maaten[128]	Bunte[29]	Engel[50]	Sorzano[192]	Cunningham[40]	Gisbrecht[71]	Liu[119]	Xie[224]	Nonato[148]	**Ours**
Number of techniques	12	6	7	14	9	9	19	14	8	6	4	28	**44**
Number of datasets	-	2	2	10	3	-	3	-	3	-	3	-	**18**
Number of metrics	-	-	-	3	3	-	-	-	1	-	1	-	**7**
Field of survey	ML	IV	ML	ML	ML	IV	ML	ML	ML, IV	IV	ML	IV	**IV**

some surveys include many techniques, but discuss these from a technical/mathematical viewpoint rather than a practical one [40], or have a more educational, rather than evaluational, purpose [192]. Visualization surveys cover much more than projections and thus cannot include in-depth evaluations [50, 119]. Van der Maaten *et al.* [128] is the closest survey to our aims. Following Fig. 3.1, we next extend this survey's workflow, by considering more DR techniques (44 in total), more datasets (18 in total), an explicit choice of datasets to cover better the variability present in high-dimensional data spaces $\mathcal{D}$, more quality metrics (5 scalar metrics and 2 visual ones), and a study of how quality depends on the projection algorithms' parameters.

Table 2 lists the projection techniques considered in this survey as well as the abbreviations we use for them.

3.2 DATASETS

Our first step (Fig. 3.1a) is to sample the space $\mathcal{D}$ of existing multidimensional *datasets* to get a representative collection $\overline{D}$ on which we evaluate projection techniques. For this, we propose a set of *traits* to characterize datasets (Section 3.2.1). Binning these traits enables us to construct $\overline{D}$ (Section 3.2.2).

3.2.1 *Dataset Traits*

The dataset traits we propose to describe $\mathcal{D}$ aim to capture aspects outlined as important for the behavior of projection algorithms in earlier surveys [40, 128, 192]. We also choose traits that are easy to understand and measure by non-specialist end-users (the audience of our work), so they can easily use them when evaluating existing techniques *vs* their own datasets. We propose the following five traits, along with sampling strategies that create classes of elements in $\mathcal{D}$ along each trait:

Type τ_D: This trait has three categorical values, *tabular*, *image*, and *text*, in line with the most frequent dataset types for which projections are used [148]. We define three classes: *tables*, *images*, and *text*, one per value of τ_D.

Size N: Number of samples in a dataset. We define three classes: *small* ($N \leq 1000$); *medium* ($1000 < N \leq 3000$); and *large* ($N > 3000$). These values are in line with typical dataset sizes used in projection evaluation papers.

Dimensionality n: Number of dimensions of a dataset. We define three classes: *low* ($n < 100$); *medium* ($100 \leq n < 500$); *high* ($n \geq 500$). Typically, the lower the dimensionality n is, the easier is the job of a projection technique. While it can be argued that this difficulty is chiefly a function of the intrinsic dimensionality (discussed next), typical end users first get exposed to, and can easily evaluate, n; in contrast, evaluating the intrinsic dimensionality is more involved, as one can define it in different ways, and also this metric can take different values in different neighborhoods of the data. Hence, we include n as a separate trait.

Intrinsic dimensionality ratio ρ_n: The percentage of principal components (of the total n), computed by PCA, needed to explain 95% of the data variance. Higher ρ_n values (in $[0, 1]$) typically tell that a projection P has difficulties in mapping the data to $\mathbb{R}^q$. We define three classes: *low* ($\rho_n \leq 0.1$); *medium* ($0.1 < \rho_n \leq 0.5$); *high* ($0.5 < \rho_n \leq 1$).

Sparsity ratio γ_n: We define $\gamma_n = 1 - \frac{u}{nN}$, $\gamma_n \in [0, 1]$, where u is the number of non-zero data values, and nN is the total number of data values in a dataset (including zero). Datasets have widely different γ_n values: Text word vectors are very sparse; tabular data with a few variables are very dense. Typically, the sparser the data, the closer are the datapoints in high-dimensional space [18, 23], so a projection P has difficulties in separating clusters in $\mathbb{R}^q$. We define three classes: *dense* ($\gamma_n \leq 0.2$); *medium* ($0.2 < \gamma_n \leq 0.8$); and *sparse* ($0.8 < \gamma_n \leq 1$).

Other traits are envisageable, such as considering data with (or without) missing values. We do not consider this specific trait, as it is hard to decide how to sample the 'lack of values' of $\mathcal{D}$ in a good, exhaustive, manner. Similarly, other trait classes are possible, *e.g.*, transaction data, time-series data, or network data for the 'type' trait. We subsume these to the 'table' class, as using too many classes would increase the (already large) evaluation effort by several factors.

Defining the above traits and their classes (bin values) is, of course, not a theoretically ideal way to reflect the distribution of all datasets in the space $\mathcal{D}$. Ideally, we would know which are the independent generative axes (traits) of this space, and how all datasets in the real world distribute along these, and derive the trait-bins by following characteristics of these distributions. However, since this information is not known, nor, we argue, can be inferred (even with significant effort), we take a different path: We choose *traits* based on data characteristics which are known, from previous surveys and DR papers, to be relevant for the behavior of DR methods; and choose trait *classes* (bins) based on the characteristics of datasets that end users will arguably meet when applying DR in practice.

Sampling $\overline{D}$ from $\mathcal{D}$ along these five traits allows us to evaluate projection techniques P on different types of datasets, aiming to answer questions such as:

- How does P work for datasets of different *types*? Is the type of a dataset important when choosing P?
- How does P scale with the number of *samples* and/or *dimensions* of a dataset?
- How does P handle data with low/medium/high intrinsic *dimensionality*?
- How does P behave for *sparse vs* non-sparse data?

3.2.2 *Chosing Datasets*

Sampling $\overline{D}$ from $\mathcal{D}$ along the trait bins introduced in Section 3.2.1 is challenging. Taking one sample per combination of intervals would yield $3^5 = 243$ different datasets, which would make the evaluation impractical, given that we next want to evaluate several tens of techniques per sample. Also, finding *real world* datasets for all these trait values is hard. Separately, we need datasets having *labeled* data, given that some quality metrics depend on this (Section 2.2). Hence, we chose to manually collect a smaller set of 18 datasets which cover well (though not fully) the aforementioned space of trait values, and we subsampled some of the larger datasets, to avoid having too long runs or too many fails in the evaluation, since not all techniques are well suited to large data. The selected datasets for this evaluation are introduced below, and are described in detail in Section 2.3. Table 4 lists their trait values.

Table 4: Selected datasets $\overline{D}$ and their trait values. See Sections 2.3 and 3.2.2.

Dataset	**Type** (τ_D)	**Size** (N)	**Size class**	**Dimensionality** (n)	**Dimensionality class**	**Intrinsic dim.** (ρ_n)	**Intrinsic dim. class**	**Sparsity** (γ_n)	**Sparsity class**
bank	*tables*	2059	*medium*	63	*low*	0.0317	*low*	0.6963	*medium*
cifar10	*images*	3250	*large*	1024	*high*	0.0706	*low*	0.0024	*dense*
cnae9	*text*	1080	*medium*	856	*high*	0.3201	*medium*	0.9922	*sparse*
coil20	*images*	1440	*medium*	400	*medium*	0.0105	*low*	0.3858	*medium*
epileptic	*tables*	5750	*large*	178	*medium*	0.2191	*medium*	0.0067	*dense*
fashion_mnist	*images*	3000	*medium*	784	*high*	0.2385	*medium*	0.5021	*medium*
fmd	*images*	997	*small*	1536	*high*	0.3073	*medium*	0.0095	*dense*
har	*tables*	735	*small*	561	*high*	0.1194	*medium*	0.0001	*dense*
hatespeech	*text*	3222	*large*	100	*medium*	0.6130	*high*	0.9993	*sparse*
hiva	*tables*	3076	*large*	1617	*high*	0.2498	*medium*	0.9091	*sparse*
imdb	*text*	3250	*large*	700	*high*	0.5790	*high*	0.9945	*sparse*
orl	*images*	400	*small*	396	*medium*	0.0006	*low*	0.9000	*sparse*
secom	*tables*	1567	*medium*	590	*high*	0.0102	*low*	0.2617	*medium*
seismic	*tables*	646	*small*	24	*low*	0.0417	*low*	0.5883	*medium*
sentiment	*text*	2748	*medium*	200	*medium*	0.8080	*high*	0.9936	*sparse*
sms	*text*	836	*small*	500	*medium*	0.7240	*high*	0.9947	*sparse*
spambase	*text*	4601	*large*	57	*low*	0.0351	*low*	0.7741	*medium*
svhn	*images*	733	*small*	1024	*high*	0.8734	*high*	0.0001	*dense*

3.3 PROJECTION TECHNIQUES

Just as we sampled the space of multidimensional datasets $\mathcal{D}$ (Section 2.3), we must now sample the space of projection techniques $\mathcal{P}$ (Fig. 3.1b). For this, we could use one of the projection taxonomies in the literature. Yet, this poses problems: There is so far no agreed 'universal' taxonomy. Different taxonomies serve different goals. For instance, Van der Maaten *et al.* [128] organize techniques on the type of *optimization* method they use; Cunningham *et al.* [40] follow a similar approach, but cover only *linear* techniques; Sorzano *et al.* [192] classify methods on *implementation* aspects (statistics-based, dictionary-based, and projection-based); Engel *et al.* [50] also classify methods on *implementation* aspects (projection-based, graph-based, and manifold learning). Finally, Nonato *et al.* [148] classify methods along eight traits (Section 3.1.2). We follow a similar approach, but use different traits, as explained next.

3.3.1 *Projection Traits*

We base our sampling $\overline{P}$ of the space of projection techniques $\mathcal{P}$ on eight traits that reflect what non-specialist users consider when choosing a technique, as follows.

Linearity: A projection is linear or nonlinear. Both types are well-covered in the literature and equally important in practice. Linear projections are easy to understand and use, but cannot capture well sample distributions spread on complex manifolds in nD. Nonlinear projections are better for such datasets, but are harder to control parameter-wise;

Input type: A projection P reads either a *distance* matrix $A = (d(\mathbf{x}_i, \mathbf{x}_j))$, $1 \leq i \leq N$, $1 \leq j \leq N$, where d is a dissimilarity function over $\mathcal{D}$, or the set $D = \{\mathbf{x}_i\}$ of high-dimensional *samples* themselves. When samples are available, one can always derive a distance matrix from them, but not conversely;

Neighborhood: A projection P claims to preserve local or global neighborhoods. Local-neighborhood methods try to preserve distances between a point and its (close) neighbors in D, which may yield better cluster separation, but lose the meaning of distances between clusters in the projected space [127]. Global methods try to preserve all-point-pair distances, which may result in more faithful projections of the high-dimensional space, but show cluster separation less well [97];

Ease of use: Number of free parameters (hyperparameters) that P exposes to the end user. More parameters give more flexibility, but finding optimal settings is harder;

Computational complexity: Algorithmic complexity of P, in big-O notation, as a function of N and n. Low-complexity methods are best for interactive visual exploration, but may have trouble in creating accurate results;

Out-of-sample: Ability to project new data based on earlier training. Useful when one wants to study dynamic datasets which add new samples over time [138, 148, 166];

Inverse transform: Ability to map low-dimensional $\mathbb{R}^q$ data to the original $\mathbb{R}^n$ space. Useful for explaining patterns in the projection [5, 58, 148, 188];

Determinism: Ability to reproduce its results regardless of random seed initialization. Useful when reproducible results are expected.

3.3.2 *Selected Projections*

Following the above, we select a set $\overline{P}$ of 44 DR techniques which cover a wide set of the end-user-relevant trait values (Section 3.3.1). As when selecting datasets to create $\overline{D}$ (Section 3.2.2), our sample $\overline{P}$ cannot cover *all* possible DR techniques. To make $\overline{P}$ as relevant as possible, we selected DR techniques that are well known, often met in literature or practice, have a readily available implementation, and can be applied to generic multidimensional datasets (as opposed to handling very specific kinds of data). This way, we maximize the likelihood that $\overline{P}$ includes most techniques of interest that a typical user will consider and want to ask questions about.

We next describe the selected techniques. Citations indicate the specific variant of a technique we used. Table 5 summarizes their trait values, including the publicly available implementations we used in our evaluation. Except the number of free parameters of each algorithm, and the implementation we used, which are self-explaining, all other traits are described in [148]. We do not detail the theoretical or algorithmic aspects of these techniques, as these are covered in earlier surveys or original papers cited below, and since we aim to evaluate these techniques from an *end user* perspective rather than from a designer's or mathematician's one. We group these techniques along two attributes, linearity and type of neighborhood, each having two values, yielding four groups. This simple taxonomy helps non-specialist users to first select an appropriate group of techniques for their problem, after which they can refine selection based *e.g.* on quality metrics (Section 3.4).

Linear and Global: Techniques that use only linear transformations and consider all samples at a time. This group includes **PCA**[97] and its variations, Incremental PCA (**I-PCA**)[174], Probabilistic PCA (**P-PCA**)[205], and Sparse PCA (**S-PCA**)[231], all of which use orthogonal transformations to derive a set of uncorrelated variables. Factor Analysis (**FA**)[97] and Fast ICA (**F-ICA**)[93] are related to PCA, but aim at uncovering latent variables not captured by existing data dimensions. Nonnegative Matrix Factorization (**NMF**)[113] and Truncated SVD (**T-SVD**)[80] use matrix factorization to find representations in lower dimensions. Locality Preserving Projection (**LPP**)[83] is an algorithm based on linear projective maps.

Nonlinear and Local: Techniques that use nonlinear functions and seek to preserve the local neighborhood for each sample. This group contains manifold learning techniques such as Isomap (**ISO**)[203] and its faster variant Landmark Isomap (**L-ISO**)[33], both of which use geodesic distances to estimate neighborhoods; Locally Linear Embedding (**LLE**)[175] and its variants Hessian LLE (**H-LLE**)[46], Modified LLE (**M-LLE**)[229] and Local Tangent Space Alignment (**LTSA**)[230], Laplacian Eigenmaps (**LE**)[17], Diffusion Maps (**DM**)[37], Manifold Charting (**MC**)[25], and Local Linear Coordination (**LLC**)[201]. Other techniques in this group are Local Affine Multidimensional Projections (**LAMP**)[95], which uses orthogonal mapping theory to build accurate local transformations; Projection by Clustering (**PBC**)[153], a fast method that represents sample similarity by proximity; Interactive Document Maps (**IDMAP**)[142], which maps data by a fast projection, then refine the result using a force scheme; and Maximally Collapsing Metric Learning (**MCML**)[72], that use convex optimization to learn a quadratic Gaussian metric. Last but not least, we have t-Stochastic Neighborhood Embedding (**T-SNE**)[127], a method that aims to maximize the probability that similar samples are placed close to each other, and which is considered a gold-standard for 2D projection; and Uniform Manifold Approximation and Projection (**UMAP**)[138], which aims to find a $\mathbb{R}^q$ fuzzy topological structure closest possible to the $\mathbb{R}^n$ topological data structure. Compared to t-SNE, UMAP produces in general more clustered results, and is significantly faster.

Nonlinear and Global: Techniques that use nonlinear functions and consider all samples at a time. Techniques in this group are Metric Multidimensional Scaling (**MDS**)[206], Nonmetric Multidimensional Scaling (**N-MDS**)[109] and Landmark MDS (**L-MDS**)[189]. Kernel PCA (**K-PCA**)[181] and Gaussian Process Latent Variable Model (**GPLVM**)[111] are nonlinear extensions of PCA that use kernel methods and probabilistic models, respectively. Gaussian (**G-RP**) and Sparse Random Projections (**S-RP**)[43]

project the original input space on randomly generated matrices. Maximum Variance Unfolding (MVU)[216] and its variations Fast MVU (F-MVU)[123] and Landmark MVU (**L-MVU**)[217] aim to unfold the data manifold by maximizing Euclidean distances between points while preserving pairwise distances in a neighborhood graph. Generalized Discriminant Analysis (**GDA**)[12], also known as Kernel LDA, is a nonlinear generalization of LDA, a supervised linear DR technique [24]. Least Square Projection (**LSP**)[154] and its faster version Piecewise Least Square Projection (**PLSP**)[155] use least squares approximations. Other methods in this class are Rapid Sammon (**R-SAM**)[156] and Fastmap (**FMAP**)[63]. Autoencoders (**AE**)[88] use neural networks to generate low-dimensional data representations that can be used as projections. Stochastic Proximity Embedding (**SPE**)[2] aim to preserve similarities between a set of related points.

Linear and Local: Techniques that use only linear transformations to reduce dimensionality on separate small neighborhoods. This includes Large-Margin Nearest Neighbor Metric Learning (**LMNN**)[215], which learns a Mahalanobis distance metric by using semidefinite programming; and Linear LTSA (**L-LTSA**)[228], a variation of LTSA [230] that uses linear mappings.

Other techniques: Besides the above projection techniques, and technique traits, several others exist. A particular one is the use of *labeled* data when computing the projection [116, 213]. Using such information helps better separating classes present in the data, which, in turn, yields better values for several of the projection quality techniques discussed next in Section 3.4. We did not include these techniques in the survey as they would be hard to compare against techniques that do not use label information (which are in the majority).

3.4 QUALITY METRICS

The third and last component of our benchmark (Fig. 3.1c) covers the projection quality metrics used to assess the selected methods $\overline{P}$ on the selected datasets $\overline{D}$. Using metrics to gauge the quality of DR methods is an established field for which separate surveys exist [114, 121, 148, 161]. Since DR is essentially ill-posed, several such metrics must be *jointly* used to assess the quality of a DR technique [71]. In Section 2.2 we describe all metrics used in detail.

We also considered other metrics for our benchmark, *e.g.*, Kullback-Leibler divergence [87], Local Continuity meta-criterion [32], Topographic Product [13], and Procrustes Measure [73]. Interpreting such metrics is harder [148] and thus provides arguably less (clear) feedback for our typical target users, so we refrained from using them.

A special class of scalar quality metrics aims to capture perceived *visual separation* of clusters in scatterplots [3, 144, 184]. Closer to our context, Tatu *et al.* [199, 200] study four such metrics on 2D scatterplots containing labeled samples. Sedlmair and Aupetit survey 14 additional metrics for the same goal [183]. Both above papers conclude that Distance Consistency (DSC) [190] (called Class Consistency Measure (CCM) in [200]), defined as the percentage of points $\mathbf{x}$ whose nearest class-center-of-mass belongs to the same class as $\mathbf{x}$, best approximates the way humans rank visual separation. More recently, ML techniques were proposed to search a large space of 2002 synthesized metrics to capture even more accurately human perception [11]. While such measures can very effectively model human perception of class separation, they cannot be *directly* used in

Table 5: Selected projection techniques for evaluation and their trait values (Section 3.3.1). In the *Complexity* column, n is the number of dimensions; N is the number of samples; i is the number of iterations (for neural network training), and w is the number of weights (for a neural network).

Projection	Linearity	Input	Neighborhood	Free parameters	Complexity	Out-of-sample	Inverse transform	Deterministic	Implementation
AE	nonlinear	samples	global	network size	$O(iNw)$	yes	no	no	Keras
DM	nonlinear	samples	local	2	$O(N^3)$	no	no	yes	Tapkee
FA	linear	samples	global	1	$O(n^3)$	yes	no	yes	scikit-learn
FMAP	nonlinear	distances	global	0	$O(N)$	no	no	yes	Vispipeline
GDA	nonlinear	distances	global	1	$O(n^3)$	no	no	yes	DR Toolbox
GPLVM	nonlinear	distances	global	1	$O(n^3)$	no	no	no	DR Toolbox
F-ICA	linear	samples	global	2	$O(n^3)$	yes	yes	yes	scikit-learn
IDMAP	nonlinear	samples	local	3	$O(N^2)$	no	no	yes	Vispipeline
ISO	nonlinear	samples	local	1	$O(N^3)$	yes	no	yes	scikit-learn
L-ISO	nonlinear	samples	local	1	$O(N^3)$	no	no	no	Vispipeline
LAMP	nonlinear	samples	local	3	$O(Nn)$	yes	yes	no	Vispipeline
LE	nonlinear	distances	local	0	$O(N^3)$	no	no	no	scikit-learn
LLC	nonlinear	samples	local	3	$O(in^3)$	no	no	yes	DR Toolbox
LLE	nonlinear	samples	local	3	$O(N^3)$	yes	no	no	scikit-learn
H-LLE	nonlinear	samples	local	3	$O(N^3)$	yes	no	no	scikit-learn
M-LLE	nonlinear	samples	local	3	$O(N^3)$	yes	no	no	scikit-learn
LMNN	linear	samples	local	3	$O(n^2)$	no	no	yes	DR Toolbox
LPP	linear	samples	global	1	$O(N^3)$	yes	no	yes	Tapkee
LSP	nonlinear	samples	local	4	$O(N^3)$	no	no	yes	Vispipeline
LTSA	nonlinear	samples	local	3	$O(N^3)$	yes	no	no	scikit-learn
L-LTSA	linear	samples	local	1	$O(N^3)$	no	no	no	Tapkee
MC	nonlinear	samples	local	2	$O(in^3)$	no	no	yes	DR Toolbox
MCML	nonlinear	samples	local	0	$O(n^2)$	no	no	no	DR Toolbox
MDS	nonlinear	distances	global	2	$O(N^3)$	no	no	no	scikit-learn
L-MDS	nonlinear	distances	global	1	$O(N^3)$	no	no	no	Tapkee
N-MDS	nonlinear	samples	global	2	$O(iN^2)$	no	no	no	scikit-learn
L-MVU	nonlinear	samples	global	2	$O(N^3)$	no	no	no	DR Toolbox
NMF	linear	samples	global	4	$O(n^2)$	yes	yes	no	scikit-learn
PBC	nonlinear	samples	local	4	$O(N\sqrt{N})$	no	no	yes	Vispipeline
PCA	linear	samples	global	0	$O(n^3)$	yes	yes	yes	scikit-learn
I-PCA	linear	samples	global	0	$O(n^3)$	yes	yes	no	scikit-learn
K-PCA-P	nonlinear	samples	global	1	$O(N^3)$	yes	yes	no	scikit-learn
K-PCA-R	nonlinear	samples	global	1	$O(N^3)$	yes	yes	no	scikit-learn
K-PCA-S	nonlinear	samples	global	1	$O(N^3)$	yes	yes	no	scikit-learn
P-PCA	linear	samples	global	1	$O(N^3)$	yes	no	yes	DR Toolbox
S-PCA	linear	samples	global	3	$O(N^3)$	yes	no	yes	scikit-learn
PLSP	nonlinear	samples	global	0	$O(N^3)$	no	no	yes	Vispipeline
G-RP	nonlinear	samples	global	0	$O(Nn^3)$	yes	no	no	scikit-learn
S-RP	nonlinear	samples	global	0	$O(Nn^3)$	yes	no	no	scikit-learn
R-SAM	nonlinear	samples	global	0	$O(iN^2)$	yes	no	no	Vispipeline
T-SNE	nonlinear	distances	local	3	$O(iN^2)$	no	no	no	Multicore TSNE
SPE	nonlinear	samples	global	2	$O(N^2)$	no	no	no	Tapkee
T-SVD	linear	samples	global	1	$O(N^2)$	yes	yes	no	scikit-learn
UMAP	nonlinear	distances	local	3	$O(iN^2)$	yes	yes	no	umap-learn

our context: Our aim is to model how well a DR scatterplot captures aspects of the *n*D data, and not how the plot is actually perceived by users for a given task.

By analogy, while the ground-truth in the comparison in [11, 183] and related work is human perception, such ground truth is the *n*D data in our case. In terms of [148], we are interested in the DR errors at the so-called 'model stage', not at the visualization stage. Hence, we cannot directly use such metrics in our case. The only exception here is the DSC metric, which relates the 2D scatterplot to the underlying *n*D data structure. However, DSC's formulation assumes that clusters are *well separated* in *n*D (for details see [190, 200]). Our benchmark, as well as the space $\mathcal{D}$, contain more general datasets which do not necessarily comply with this constraint.

Other scalar metrics exist and, as with any work, we had to make choices of which aspects we leave out of our study. For instance, *stability* [148] is a relevant metric which quantifies how much a projection changes upon changes of either its parameters or changes in the input data. Ideally, projections should be smooth functions of such changes, so small data and/or parameter changes imply only small changes in the visual result. However, formally defining stability in both these senses is hard, and quantifying it over the entire spectrum of data- and parameter-changes is even harder. Hence, our choice to leave stability out of the evaluation.

3.5 MEASUREMENT METHOD

Following Equation (2.3), we need to evaluate our metrics $\overline{M}$ on all projection techniques in $\overline{P}$ applied to all considered datasets in $\overline{D}$ (Fig. 3.1d). In this process, we must also consider the *free parameters* that different projection techniques expose (Table 5). Obviously, the quality metric values will depend on these parameter choices. We next discuss the parameter search process used to handle this.

First, we need to define what is an *optimal* projection with respect to our quality metrics. For this, we aggregate the considered metrics to yield

$$\mu = \frac{1}{5}\left(M_{nh} + M_t + M_c + (1 - M_\sigma) + M_s\right) \tag{3.1}$$

where M_{nh}, M_t, M_c, M_σ, and M_s are the scalar neighborhood hit, trustworthiness, continuity, stress, and Shepard goodness metrics in Section 3.4. As all these metrics range over $[0, 1]$, and we consider them equally important, they have identical weights in Equation (3.1). Equation (3.1) does not consider local metrics (Section 2.2.3) since these yield images meant to be qualitatively assessed, and also can not be always ranked (ordered) in terms of one image being globally better or worse than another one.

Let now $\pi_i \in \Pi_i$ denote the free parameters of a projection technique $P \in \overline{P}$, where Π_i indicate their allowable ranges. Let $\mu(P(D, \pi_i))$ be the aggregated quality of P run over dataset D with parameter values π_i. How can we then define what the "optimal" quality for a technique P is (which we need next to compare techniques)? We propose for this two solutions, as follows.

Dataset-wise view: For each $D \in \overline{D}$, we compute the optimal projection $P^{opt}(D)$ by doing a grid search that maximizes μ (Equation (3.1)) over the ranges Π_i of all parameters π_i of P, *i.e.*,

$$P^{opt}(D) = \underset{\pi_i \in \Pi_i}{\arg\max}\, \mu(P(D, \pi_i)) \tag{3.2}$$

The exact details of the optimization process, including the parameters π_i and their ranges Π_i for each technique P, are given in Appendix B.1. During this process, we were

careful to reset the random seeds used by non-deterministic optimization algorithms, *e.g.*, t-SNE, at the beginning of each subsequent execution of the same algorithm. This way, the results we report in here can be replicated.

We next denote the quality of $P^{opt}(D)$ by $\mu^{opt}(P, D)$, and the parameter set realizing this quality by $\pi^{opt}(P, D)$ respectively. The values $\mu^{opt}(P, D)$ allow us to study how well a technique P performs over the different types of datasets in $\overline{D}$, and also how well different techniques perform for the same dataset. Key to this is that the techniques are *optimized independently per dataset*. This allows seeing the "best" that a given technique can yield for each dataset. Also, by studying the distribution of optimal parameters $\pi^{opt}(P, D)$ over $\overline{D}$, we can assess how much parameter tuning a technique P actually needs in practice. The results of using the dataset-wise view on our benchmark are discussed in Section 3.6).

Projection-wise view: The dataset-wise view chooses a separate optimal parameter set $\pi^{opt}(P, D)$ for each dataset D. Clearly, this expensive grid-search cannot be done in routine practice. Rather, users like to have so-called parameter *presets* when applying a technique P on *any* dataset. The projection-wise view measures quality in this way: For each technique $P \in \overline{P}$, we choose as preset the parameter-set $\pi^{preset}(P) \in \pi^{opt}(P, D)$ that yields the best quality μ most times (statistical mode) over all datasets in $\overline{D}$. The projection-wise view allows comparing projection techniques more globally, *i.e.*, seeing how they perform with respect to each other when no per-dataset parameter tuning takes place. The results of using the projection-wise view on our benchmark are discussed in Section 3.6).

3.6 RESULTS

After evaluating our benchmark, we obtain a multidimensional dataset consisting of five quality metrics (plus the aggregated one, see Equation (3.1)), measured for 44 projections, each run over 18 datasets, and additional values for optimal parameters – in total, over 5000 measured values. We next present and discuss ways to (visually) explore this measurements dataset to gain insights on the tested projections, and answer different questions on various levels of detail.

How good are projections, and for which data? To answer this, we use the dataset-wise view (Section 3.5). The table in Fig. 3.2 has rows for DR techniques P and columns for datasets D. Each cell shows the optimal quality value $\mu^{opt}(P, D)$, color coded by a sequential colormap. Grey cells show techniques which could not complete the projection of the respective datasets (crashing or hanging).

Scanning Fig. 3.2 along *rows* shows how much the optimal quality of a given projection varies over the studied datasets. For instance, we see that the projection-set starting with GLPVM and ending with LLE has quite similar (and high) optimal qualities over all datasets — that is, Fig. 3.2 shows a relatively dark-green compact block of cells starting with the GLPVM row and ending with the LLE row. In contrast, if we focus in Fig. 3.2 on the block spanned by the PCA-class projection rows (starting with PCA and ending with S-PCA), over all columns, we see little variation of colors along columns and more variation along rows, respectively. Hence, the PCA-class projections have quite similar optimal qualities for the given dataset, but the optimal qualities vary as a function of the dataset itself. Another salient pattern is given by N-MDS. While N-MDS only failed for a single dataset (*spambase*, grey cell on N-MDS row in Fig. 3.2), the respective row

shows a higher error for N-MDS for basically all datasets and compared to most other projection techniques.

Scanning the figure along columns shows which are the best projections for a given dataset. For instance, we see that the earlier-mentioned set of projections starting with GPLVM and ending with LLE, and also all PCA variants, yield very good quality for the *orl*, *secom*, and *seismic* datasets. This can be explained by the fact that these datasets have a low intrinsic dimensionality (see Table 4) and these projection techniques handle very well such data. In contrast, *sentiment* has the second-highest intrinsic dimensionality of all datasets, and we see that it also yields relatively lower optimal qualities for all projections. Overall, t-SNE, UMAP, IDMAP and PBC perform well on average for most datasets. PLSP, LLTSA, LPP and GDA perform poorly. The PCA variants perform reasonably well for most datasets.

To better understand quality, we next explore how easy is to obtain optimal values for it, and how quality depends on parameter values and dataset types:

How easy is to obtain optimal quality? The rightmost four columns of Fig. 3.2 show the standard deviations of the optimal parameters $\pi^{opt}(P, D)$, computed over all datasets D treated by every projection P, normalized to the interval $[0, 1]$, depicted by a heat colormap (darker = higher standard deviation). Empty cells correspond to techniques that have less than four parameters (see Appendix B.1). Good projections are those which yield high optimal quality values (green cells along their rows) *and* achieve this with little parameter tuning (light cells in the parameter columns, if they have any parameters). For instance, ISO is better than L-ISO as it achieves overall the same maximal quality, but requires less parameter tuning (0.27 *vs* 0.47 variance). PBC is better than NMF as it achieves slightly higher maximal quality and requires less parameter tuning for that. We also see that the four overall best performing techniques (t-SNE, UMAP, IDMAP and PBC) require some tuning effort over most parameters to yield optimal results. Of the techniques that do not require parameter tuning, PCA is the best performing, albeit with lower quality than that of the best projections.

How does quality depend on parameter settings? To give more insight into this, inside each cell, a four-bin histogram shows how many runs, with different parameter values, done during the grid-search process over the parameters π_i, achieved a quality μ in the ranges $[0.0, 0.62)$, $[0.62, 0.75)$, $[0.75, 0.87)$, and $[0.87, 1.0]$ respectively. These seemingly arbitrary ranges were selected because most of the data is located above the 0.5 threshold, so creating uniformly divided bins would not produce the desired outcome. Note that above 0.5, the bins are divided uniformly. Histograms with long *leftmost* bars indicate that, for the respective projection technique and dataset combination, most runs (parameter values) yield bad quality (undesirable situation). Histograms with long *rightmost* bars indicate that the respective technique-dataset combination achieves good quality for most parameter combinations (desirable situation). For instance, K-PCA-P has overall one long leftmost bar for all datasets, so it yields poor quality for most of the tested parameter combinations. There is no technique that shows long rightmost bars for all datasets. Hence, it is very hard to consistently obtain high quality for all datasets by parameter tuning. Histograms having several *non-zero* bars indicate methods where parameter tuning is crucial to obtain good performance, *e.g.*, LLE. Histogram shapes also depend on the *dataset*: For the *svhn* dataset column, most histograms are 'spread out', indicating that this dataset is hard – it requires more parameter tuning than other datasets to get good quality. For *imdb*, most histograms have one long leftmost bar,

Table 6: Correlation between dataset traits and optimal quality values.

Intrinsic dim.	Sparsity ratio	Dimensionality
-0.630390	-0.289365	-0.090593

telling that this dataset is *very hard* to project regardless of parameter tuning.

How does quality depend on* dataset type*? To answer this, Table 6 presents the correlation between dataset traits (Section 3.2.1) and the optimal quality values in Fig. 3.2. Several findings follow:

- *Intrinsic dimensionality* ρ_n is the trait that most influences quality μ, with average correlation -0.63. Hence, data with high intrinsic dimensionality is hard to project by all studied techniques;
- *Sparsity ratio* γ_n follows, with a correlation of -0.29 to quality, indicating that sparser datasets are harder to project well;
- *Dimensionality* n has a very low correlation of -0.09, barely affecting quality. The same holds for dataset type τ_D and size N. Hence, one should not worry in practice about these traits when choosing a projection technique that should yield high quality. Of course, such traits influence other aspects, such as computational speed (discussed later in Section 3.6).

How good are parameter-preset projections? Using parameter presets is desirable for typical end users. We examine how well projections perform using presets with the projection-wise view (Section 3.5). Figure 3.3 shows a table with the same layout as the dataset-wise view (Fig. 3.2). However, we now compute the quality μ^{preset} using the *same* preset parameters $\pi^{preset}(P)$. The four rightmost colums show the preset values. Comparing this image with Fig 3.2, we see how quality drops overall. For more insight, we show the quality loss $\mu^{opt} - \mu^{preset}$ separately in Fig. 3.4. Figure 3.3 answers several practical questions:

- The four right columns shows which parameter *presets* one can use for each projection technique to get overall good quality, regardless of the dataset;
- Comparing *rows* allows seeing how two projections fare, quality-wise, when using presets. Overall, t-SNE, UMAP, IDMAP and PBC are the best-performing techniques in this sense;
- Comparing *columns* shows datasets which are 'easy' (*e.g.*, *orl*, *secom*, and *seismic*) or 'hard' (*e.g.*, *cnae9*, *imdb*, and *sentiment*) to project well when using presets. When one has a concrete dataset, one can find the benchmark dataset sharing similar traits (see table at the bottom of Fig. 3.3) and infer how a given projection would perform on it, or which is a good projection for this kind of dataset, using presets. This allows a first rough selection of good projection candidates.

Which projections perform similarly well? The dataset-wise and projection-wise views convey many details on the specific behavior of a projection as function of the datasets and parameter tuning. However, this amount of detail can be overwhelming for the end user interested in comparing projections on a high level. Moreover, we do not have insights into the behavior of projections *vs* their raw, non-aggregated, quality metrics. For this, we consider each projection technique P attributed by the values of its

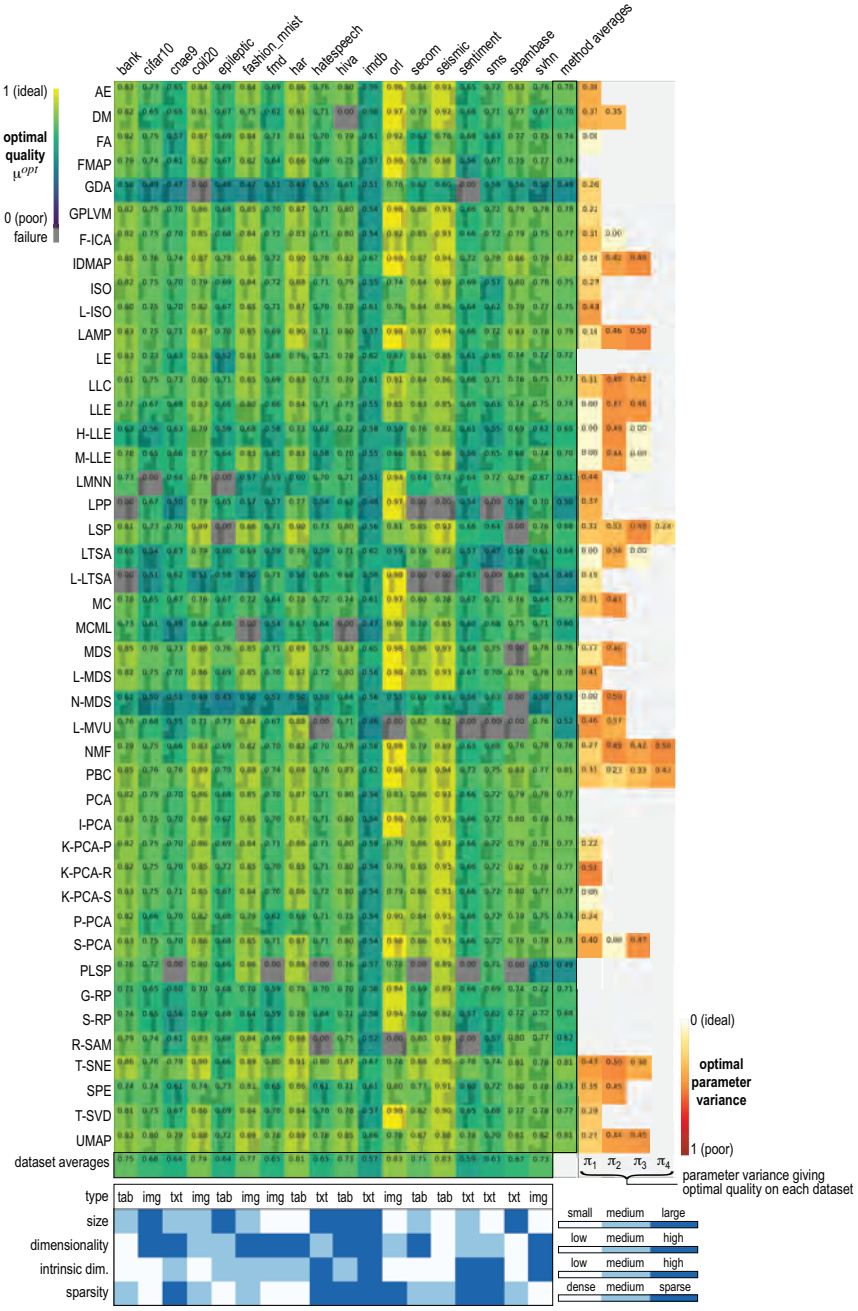


Figure 3.2: Dataset-wise view showing optimal quality per dataset (columns), all projections (rows). Histograms in each cell indicate number of runs divided into four quality bins, the bottom table shows dataset trait values. See Section 3.6. Parameters π_i are discussed in Appendix B.1.

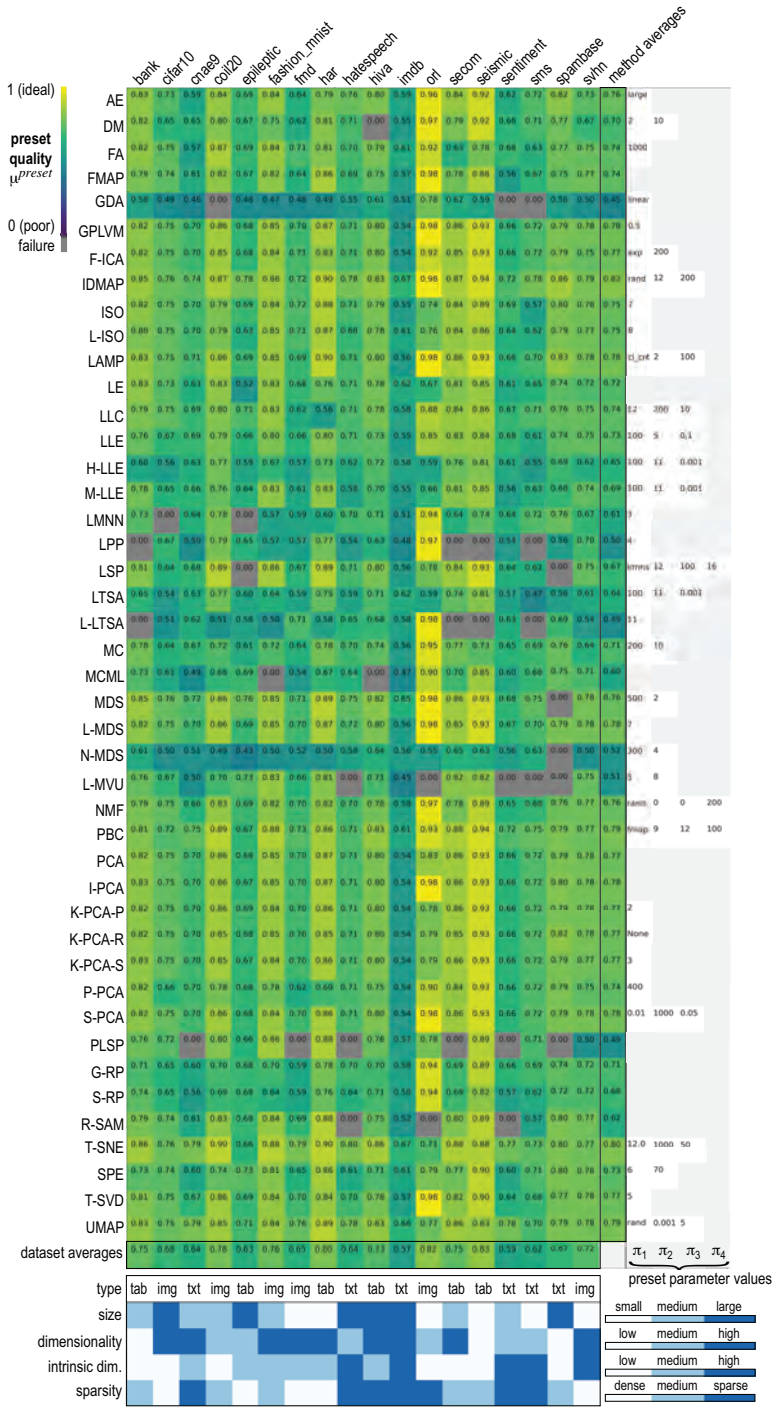


Figure 3.3: Projection-wise view (Section 3.6) of quality per dataset (columns), all projections (rows) for preset parameters. Appendix B.1 discusses parameters π_i.

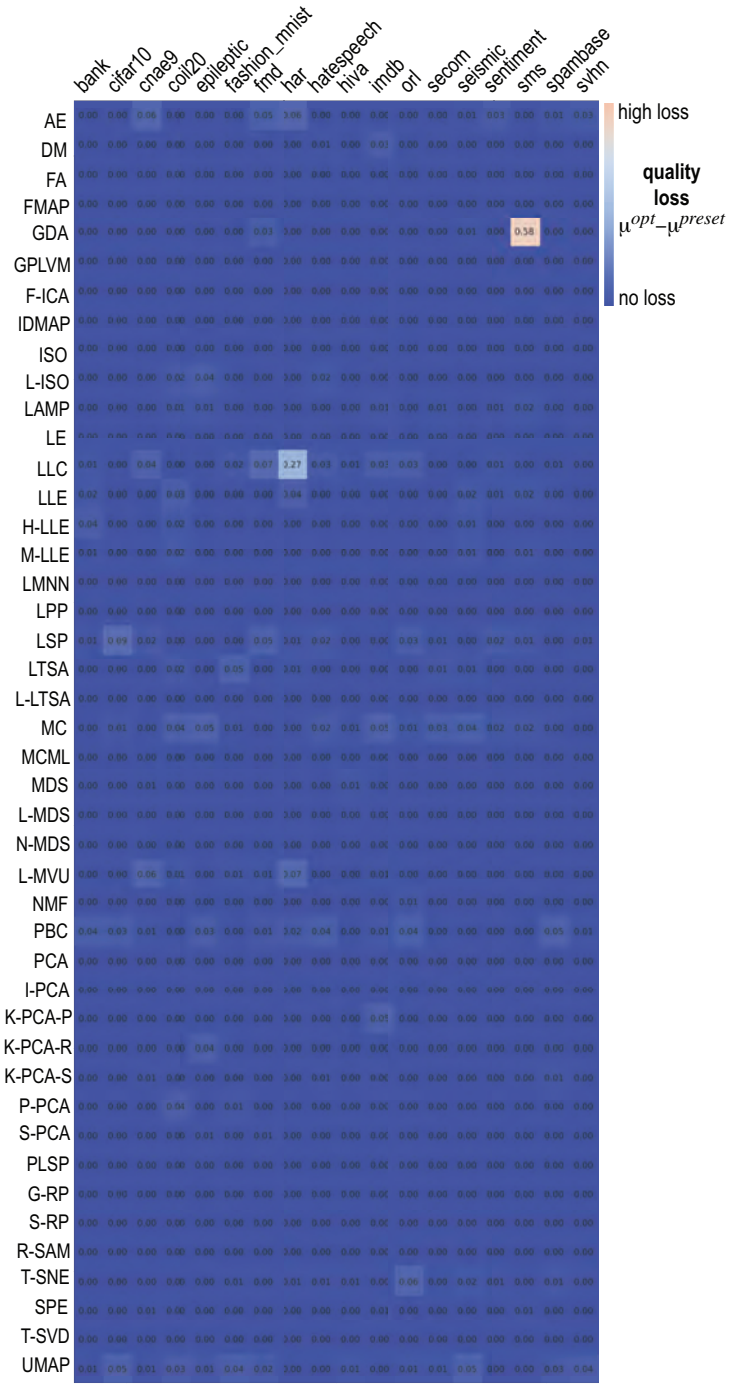


Figure 3.4: Quality loss $\mu^{opt} - \mu^{preset}$. See Section 3.6.

five quality metrics (Equation (3.1)), averaged over all datasets D. We next project this set using MDS and color the resulting scatterplot by the average quality μ (Fig. 3.5a). Similar, but more elaborated designs, have been used to compare projections, [41, 157]. This 'projection of projections' map shows how similar all techniques are from the perspective of all raw quality metrics over all datasets. We see a clear gradient of the average quality μ ranging from N-MDS and GDA (poorest) to UMAP, t-SNE, PBC and IDMAP (best). We also see that methods in the same family perform relatively similar, *e.g.*, the PCA variants. To explain the direction orthogonal to the color gradient, we color points (projection techniques) in turn by each metric and look for patterns. We find that this direction maps well the stress M_σ. These insights depend, of course, on the quality of the MDS projection used. To choose a good projection for this dataset, we could find DR methods that score well on datasets having similar trait values ($n = 5$, $N = 44, \gamma_n = 1, \tau_D = tabular$) following our analysis in Section 3.6. We do not take this path here since this dataset is very small and simple, and thus arguably projectable well by established methods such as MDS. To gain more confidence, we redo the plot using t-SNE (Fig. 3.5b). The orientation of the average quality (color gradient) and stress axes differs, but the overall pattern is very similar. Using these plots, users can compare projection techniques from the perspective of overall quality (to choose optimal ones), but also can choose techniques which behave similarly to a given technique of interest.

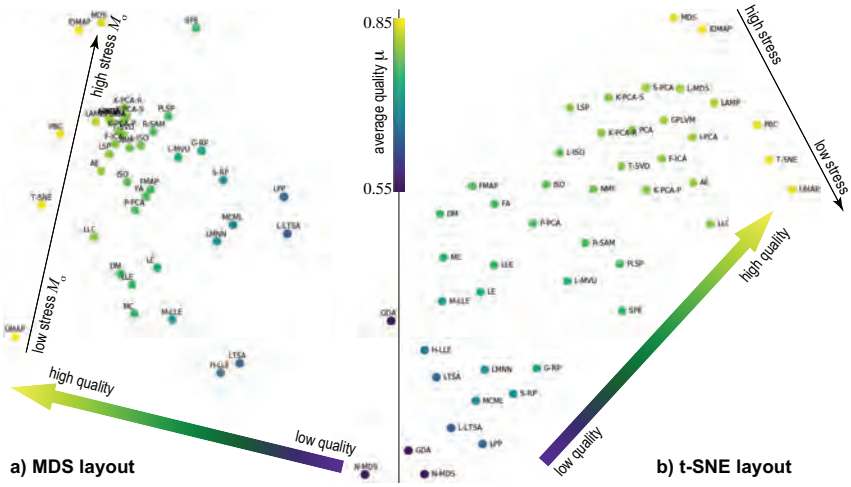


Figure 3.5: Projection of projections based on five quality metrics. Color shows average quality μ. See Section 3.6.

Detailed study of selected good projections Several of our analyses so far point out that the top-four quality projections are UMAP, t-SNE, PBC and IDMAP. We now analyze these in more detail, from the perspective of speed (Section 3.6) and the way they distribute their errors over the 2D space (Section 3.6). These extra insights can help users selecting a best technique from this top-four set.

How fast are the best projections? We measure the speed of the four selected projections on synthetic Gaussian datasets for varying number of dimensions n and observations N. We sample n ranging from 50 to 1,000, and N ranging from 500 to

50,000, with 30 samples each, yielding 900 datasets that we next project and time. Figure 3.6 shows the results. Note that the four color scales correspond to different time ranges, as the four techniques have very different speeds. Normalizing all colors within the same range would suppress seeing interesting variations of the speed *vs* the parameters n and N. Hence, we chose to normalize colors per projection, and rely to annotations to convey the different time scales. We see that UMAP and PBC are almost two orders of magnitude faster than t-SNE and IDMAP. Color gradients tell us that the dimensionality n affects speed more for PBC and t-SNE than for UMAP and IDMAP. For the latter two, the sample count N affects speed more. Also, we see that t-SNE's color gradient is less smooth, being 'punctured' at a few places by outliers such as the bright yellow one (Fig. 3.6, red surrounding marker). These indicate combinations of n and N for which t-SNE took significantly longer than for similar input values, and are due to the stochastic nature of the algorithm itself. In contrast, the patterns exhibited by UMAP, PBC, and IDMAP show a smoother variation of speed with n and N.

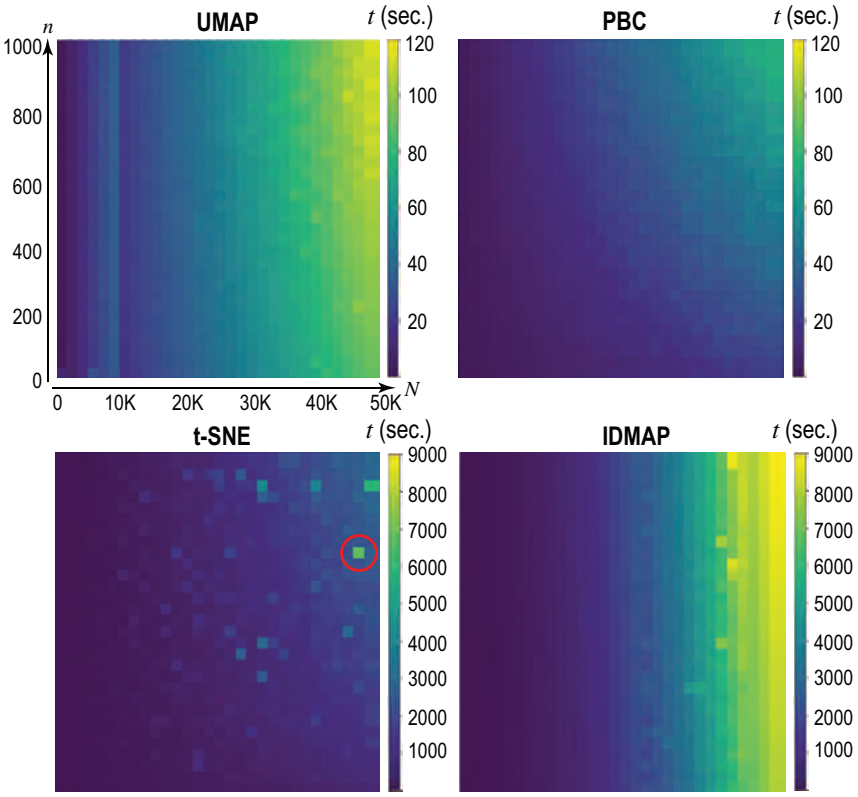


Figure 3.6: Running times for UMAP, PBC, t-SNE and IDMAP for synthetic Gaussian datasets with n dimensions, N samples. See Section 3.6.

How do projections spread their errors? We now analyze in detail how the four best techniques spread their errors over the projection $P(D)$ using Shepard diagrams and local metrics (Section 3.4). For brevity, we select a subset of four datasets, and run the four selected techniques on them using their parameter presets (Fig. 3.3). The selected

datasets represent each type of data considered in this study, namely *text* (*cnae9*), *tables* (*har*), and *images* (*coil20*, *fashion_mnist*).

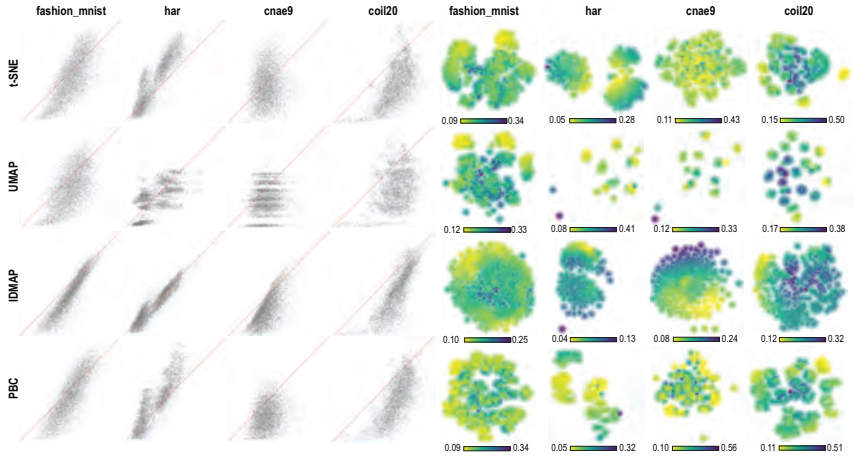


Figure 3.7: Left: Shepard diagrams for the analyzed four projections and four datasets. The x and y axes map inter-point distances in $\mathbb{R}^n$, respectively $\mathbb{R}^2$. Right: Average local errors M_a for the same datasets and projections. See Section 3.6.

First, we use Shepard diagrams (Section 2.2.2) to see how well the four techniques preserve high-dimensional distances (Fig. 3.7 left). Overall, we see that IDMAP preserves distances better than the other three techniques. At the other end, UMAP creates the most complex pattern, including both compressing and stretching distances from $\mathbb{R}^n$ to 2D. PBC and t-SNE create quite similar patterns. This is quite interesting, as it tells that we can use PBC to get very similar results to t-SNE, and PBC is about two orders of magnitude faster (Fig. 3.6).

Next, we show the actual projection scatterplots (Fig. 3.7 right), colored by the average local error M_a (Section 2.2.3). For each scatterplot, we color code M_a using a low (yellow) to high (purple) colormap. Per-scatterplot minimal and maximal M_a values are shown under the plots. We obtain several insights:

Emerging patterns: We see that the visual patterns formed by t-SNE and PBC are quite similar, in line with the earlier-detected similarity of their distance patterns in Shepard diagrams (Fig. 3.7 left). In contrast, IDMAP creates less well-separated visual clusters than all other three techniques, while UMAP creates more separated visual clusters. However, we should note that, without additional information on the ground-truth (nD data), the presence or absence or well-separated clusters in the projection is not an indication of the projection's accuracy.

Errors correlate with datasets: Looking at the M_a extrema, we see that all techniques find *fashion_mnist* to be the hardest to project, followed by *coil20*, *cnae9*, and *har*.

Error correlation with techniques: Overall, IDMAP produces the lowest errors. The other three techniques however cannot be decisively ranked, as they sometimes produce higher, and sometimes lower, errors than each other depending on the dataset. Moreover, achieving a higher pattern segregation typically implies higher M_a errors, compare,*e.g.*, t-SNE *vs* IDMAP (*har* or *fashion_mnist* datasets). Hence, M_a should not be used as a

discriminative tool for comparing projections: when studying a projection computed by a given technique, M_a is most useful for finding which scatterplot points are best (worst) projected. For a similar use-case of projection-error color-coding, see [85]. This task is discussed further below.

Error distribution: All techniques generate quite similar distributions of error values (over the error range) for all datasets, with typically few high-error points. Lowest-error points (yellow) occur most often close to the scatterplot boundaries, which has also been observed for different datasets and projections earlier [133]. In contrast, high error points (purple) appear at very different locations as a function of the technique and dataset. Hence, to actually trust a given projection, one should always (be able to) inspect such errors.

Summarizing the above, we see that t-SNE and PBC offer the best overall quality in terms of producing low errors on average, good segregation of similar point-groups (clusters), and few high-error points.

3.7 DISCUSSION

Typical surveys of projection methods propose taxonomies that cover such methods and help readers understand their underlying algorithmics and finding technically-related methods. Typical papers introducing new projection techniques present these, and offer (usually quite limited) qualitative and, sometimes, quantitative comparison with a few other techniques. Our survey covers quite different material addressing different, more practical, goals. We next discuss several observations we made during this work.

Benchmark: We present, to our knowledge, the first workflow for quantitatively evaluating projection techniques 'in the large'. For this, we describe high-dimensional data along five traits, and propose a representative sampling thereof using 18 real-world datasets of widely different dimensionality, size, type, intrinsic dimensionality, and sparsity. We next select 44 projection techniques which include, arguably, all well-known algorithms in the literature. We evaluate these techniques on these datasets along five quality metrics. In contrast to all similar evaluations so far, we study quality variation as a function of (a) the dataset traits, and (b) algorithm parameters. The entire benchmark (datasets, measurements, source code for techniques and measurement tools) is public [57], being the first such benchmark in the dimensionality reduction field. The entire workflow is implemented in Python. Specific projection implementation details are given in Appendix A.1, Table 29.

Best-quality projections: Our studies showed that t-SNE, UMAP, PBC, and IDMAP yield the best quality *vs* the considered metrics and over the considered datasets, when using preset parameters. Our parameter analysis also shows that these techniques yield high quality quite consistently when their parameters are tuned. We also provide parameter presets and show that using these decrease the optimal quality of the studied projections only slightly. All in all, this tells end users that choosing one of these four techniques, with its respective parameter presets, can consistently deliver good quality.

Similar-quality projections: We compare all 44 studied projections from the perspective of all 5 quality parameters. Our results show that the "space" of projection techniques can be easily ordered, from low to high quality ones, and that the notion of

average quality (using the 5 proposed quality metrics) does make sense—see smooth color-coded average quality gradient in Fig. 3.5. This helps end users to see which projections behave similarly quality-wise, supporting trade-off scenarios, when one wants to swap a technique for a similar-quality one that has, *e.g.*, a more robust, or faster, implementation.

Refining decisions: We analyze the top-four best quality techniques from the additional viewpoints of speed, distance preservation, and error spread over the 2D space. Our results show that the four techniques are quite different, even if their scalar (aggregated) quality metrics are quite similar. We discover that UMAP and PBC are about two orders of magnitude faster than t-SNE and IDMAP. However, UMAP has the worst distance-preservation pattern of the four. This offers directly actionable ways for end users to select a suitable projection from this set of four depending on their desires regarding speed and/or distance preservation.

Limitations: Densely covering the huge space of dataset types, projection techniques, algorithm parameters, and quality metrics is definitely very hard. Our work so far represents only a limited *sample B* of this space (Section 3.1). For instance, one could consider more datasets, traits, trait classes, quality metrics, or consider more runs of the considered projections, to account for those which have a non-deterministic behavior. An interesting avenue is to generate synthetic datasets that sample the desired dataset traits in a user-controlled manner. Doing this would allow a richer, and more automated, evaluation. However, how to suitably construct such a controlled dataset generator, able also to generate labels for well-separated point groups (needed for the visual assessment of projection results and computing the neighborhood hit), is not a trivial question, hence one that we consider for a significant future-work iteration.

However, our sample is considerably *denser* than other similar samples (evaluations) present in the literature, in all the considered aspects (datasets, parameter values, quality metrics, and number of studied projection techniques). Hence, we argue that our work is a necessary (but definitely not final) next step from current state-of-the-art in the quest of quantitatively evaluating the projection landscape.

We make all our results (methodology, data, code, measurements) open and public [57], so *B* is a 'live benchmark' that will grow as us, or others, will add datasets, techniques, and metrics to it. This way, coverage can increase over time with incremental efforts, sparing professionals from the very large effort required to set up such work from scratch. Concrete directions in which we plan to extend this work include (a) considering more dataset traits (Table 4), such as amount and type of noise; and (b) adding visual quality metrics to quantify the *perceived* quality of projections for given tasks, *e.g.*, class separation [3, 183, 199], and metrics for the robustness of projection to noise.

3.8 CONCLUSION

This chapter presents a survey of multidimensional projection techniques from the perspective of end users interested in understanding how specific algorithms, and their parameter settings, perform on specific types of high-dimensional datasets. For this, we proposed a methodology for constructing a benchmark that includes 44 techniques (including various combinations of their parameter values), 18 datasets, and 7 quality metrics. We propose an automatic way to evaluate this benchmark, and also several visualizations to analyze the gathered data. Our main contribution is making the methodology, benchmark, and related artifacts (datasets, techniques, metrics, visualizations, related

code) publicly open, so interested researchers can study these results but also contribute to enrich the benchmark. Additionally, our current evaluation of the benchmark can be used to choose projection that score best on any of the evaluated criteria, similar to each other, or on global average quality, with t-SNE, UMAP, PBC, and IDMAP being the top-ranked ones in the latter respect.

Many extensions are possible based on the current foundation. First, given its open source nature, our benchmark can be easily enhanced by adding more techniques, metrics, and datasets. In particular, adding the many metrics proposed by recent approaches using machine learning [11] is a low-hanging fruit. In this process, the size and dimensionality of the collected evaluation datasets will also grow. Hence, we will consider new visualization methods to explore the gathered data to better answer concrete end-user questions, such as why do certain techniques behave similarly; which parameters of a given technique most strongly affect a given quality metric; and which techniques are best suited to project datasets having certain traits. Last but not least, coupling prrojection metrics measured on real-world datasets to gauge which technique is actually better for which VA task, following up on [148] is an important potential extension. With these extensions, we hope that ours, and others' contributions, will make the benchmark grow to be a useful 'live' resource for the infovis and Machine Learning communities at large.

In the next chapter, we present in detail the methodology used to create this survey and explain the choices made.

4 SURVEY METHODOLOGY

4.1 INTRODUCTION

As explained in Chapter 1, multidimensional projections support many tasks dealing with high-dimensional data, such as *analysis* (finding interesting patterns in the data, *e.g.*, clusters, subspaces, or outliers); *simplification* (reducing the number of dimensions needed to capture the data structure); *prediction* (classification or regression tasks executed on the data samples); and *visualization* (presenting the data for exploration or communication via 2D or 3D scatterplots) [89, 100, 119, 198].

As also introduced in Chapter 1, but not elaborated further, different audiences have different requirements for DR algorithms. *Practitioners* want to apply DR to solve their specific analysis, simplification, prediction, or visualization tasks on their data. Their main question relates to how to find the DR technique *implementation* that optimally covers their requirements, including adding such algorithms to their data-processing pipeline. *Researchers* that develop novel DR techniques need to compare their (new) technique with existing ones to demonstrate its added value, and next want to share it with an as wide as possible public.

Having worked in both practitioner and researcher roles in various teams for over 10 years, we have observed several challenges, which we summarize by two key questions, as follows:

- Q_P **(Practitioners):** How to choose the best DR algorithm implementation for my context from the wide set of options available in the public arena?
- Q_R **(Researchers):** How to compare my new DR algorithm against the existing ones, and next share it with as many other practitioners and/or researchers as possible?

Both questions can be answered in many ways, and using many instruments, *e.g.*, surveys, benchmarks, and open-source repositories. One such example is the quantitative survey of projections presented in Chapter 3,

At a meta level, we ask ourselves: How to answer these questions *efficiently* and *effectively*? For practitioners, the search space (of existing DR algorithm implementations) is huge. How to approach the search process, starting from one's context-specific requirements, to find as quickly as possible the best DR techniques that fit these requirements? For researchers, the effort of developing new DR techniques is already large. How to ensure, with minimal effort, that the developed techniques are indeed better (and if so, how to measure this) than existing ones?

In this chapter, we examine both above questions in a systematic way. We identify the *workflows* that typical practitioners and researchers follow when answering these questions. Next, we identify available *instruments* to complete each step of these workflows, and discuss the challenges and limitations we observed when applying these instruments.

This chapter is based on publication [60].

In particular, we describe the architecture proposed and used in the DR evaluation benchmark presented in Chapter 3, that is generic and extensible in terms of datasets, DR algorithms, quality metrics, and visualizations. This both informs the interested reader on how we actually set up the quantitative evaluation described in Chapter 3, but also helps the practitioner in extending the respective evaluation or designing similar evaluations. Finally, we discuss ways forward for the community of practitioners and researchers interested in applying, respectively developing, DR algorithms.

4.2 BACKGROUND

An *implementation* of a projection technique P is a reusable software component, *e.g.* executable or library, that computes the function P.

Two types of requirements come into play when answering Q_P and Q_R, as usual when engineering software systems:

- *functional requirements* describe properties of the DR *technique* itself. These include the type of data the projection accepts (nD samples or a distance matrix), whether P is deterministic or stochastic, linear or non-linear, global or local, quality (measured by quality metrics), computational and memory scalability, and out-of-sample ability. Such aspects can be usually inferred from the technique's description;

- *non-functional requirements* describe properties of *implementations* of DR techniques, *e.g.* ease of use, documentation, portability, third-party software components and programming language needed, and interoperability with other toolkits. Finding these typically requires one to study specific implementations of P.

The questions Q_P and Q_R listed in Section 4.1 have preoccupied both practitioners and researchers, increasingly more in the last decade, when many DR techniques have emerged in the literature. Several sources of information are typically used to answer them. We rank these sources based on how strongly they support answering Q_R and Q_P, on an ordinal scale ranging from '- -' (least) to '++' (most), as follows.

Papers (Q_R : ++; Q_P : −−): Technical papers describing DR techniques are the prime information source for researchers aiming to understand and/or extend such techniques. Papers discuss functional aspects of the presented techniques well, but comparisons with other techniques are in general limited to a few. Non-functional aspects and *implementation* details are less thoroughly touched upon in papers, which leaves Q_P largely unanswered.

Surveys (Q_R : +; Q_P : +): Surveys compare (tens of) projections and consider more functional aspects (*e.g.*, metrics) than technical papers (see *e.g.* [50, 56, 128, 148, 192]. Surveys offer a very good way to choose *techniques* based on their functional properties. Yet, surveys rarely discuss how to choose specific implementations of these techniques, and also discuss less non-functional aspects.

Benchmarks (Q_R : −; Q_P : ++): Benchmarks gather concrete datasets and projection/metric implementations to help both practitioners and researchers to compare *practically* DR techniques against each other. They also help replicability and are very common in other fields of computer science, *e.g.* machine learning [135] or medical imaging [140, 172]. However, benchmarks are rare in the DR community. Three notable

recent benchmarks are Espadoto *et al.* [56] (18 datasets, 44 projection techniques, and 7 metrics. See Chapter 3); Vernier *et al.* [209] (focus in dynamic DR – 10 datasets, 11 techniques, 12 metrics); and SmallVis [139] (focus on t-SNE, UMAP, and LargeVis [198]).

Frameworks (Q_R : −−; Q_P : ++): Frameworks are collections of DR technique implementations designed, documented, and coded for reusability. They come as libraries, *e.g. scikit-learn* [1], Tapkee [117]; and turnkey systems, *e.g.* MATLAB (which provides out-of-the-box implementations of PCA, Factor Analysis [97], NMF [113], MDS [109, 206] and t-SNE [127]), ProjectionExplorer [95], and VisPipeline [96], which provide end-to-end tools for interactive exploration of projections. An extreme model of frameworks are code bases that implement a single technique, *e.g.* Van der Maaten's t-SNE [127], dt-SNE [166], and Espadoto's deep-learning DR technique [52]. While frameworks best support Q_P, finding which implementations (in which frameworks) best match a practitioner's set of requirements still requires significant manual trial-and-error testing of the DR implementations they provide.

From the above, we see that the search space for Q_P and Q_R is large and heterogeneously structured. This affects both *practitioners* – it is not evident where to start searching, and how to systematically search; and *researchers* – there's no unanimously accepted answer to what to compare against, what to compare on, how to compare, and how to report the results; also, researchers face the question on how to best share their results with others, *e.g.*, simply release their code or take the time to integrate it with some framework. Hence, we find a salient gap between DR research and practice. Our aim next is to provide insights on how to fill this gap with a low effort, and where the largest unanswered challenges reside.

4.3 OPERATIONAL WORKFLOWS

We identify two related, but not identical, workflows that practitioners, respectively researchers, follow to answer their respective questions Q_P and Q_R stated in Section 4.1. We inferred these workflows both from studies of existing papers and surveys in the DR literature, and from our own experience with answering both Q_P and Q_R in practice. Concerning our experience, we have studied 46 actual DR implementations. These implementations, and their main functional and non-functional aspects, are listed in Table 7. Additional functional aspects of these techniques, such as complexity and quality are provided in recent surveys [56, 209].

Figure 4.1 depicts the steps of both these workflows, which we detail next. Colored dots in the middle table show which information sources (surveys, papers, benchmarks, or our own analysis in Table 7) are mainly used to support every workflow step. For each step, we next discuss the main questions asked by practitioners and researchers, outline how these can be answered, and which are the challenges we observed in doing this.

4.3.1 *Practitioner workflow*

A. Search techniques. *Where do I start searching?* Starting points for this search are, obviously, technical papers on DR techniques and, more broadly, surveys thereof [56, 148]. Additional search sources are conference presentations, blogs, and peer input. This search typically delivers a (large) subset of potentially suitable DR *technique* candidates (but usually no specific *implementations*).

Table 7: Studied DR technique implementations with their functional and non-functional aspects listed.

Projection	Linearity	Input	Neighborhood	Parameters	Out-of-sample	Deterministic	Implementation
Autoencoder [88]	NL	S	G	network size	yes	no	Keras
Diffusion Maps [110]	NL	S	L	2	no	yes	Tapkee
Factor Analysis [97]	LIN	S	G	1	yes	yes	scikit-learn
Fastmap [63]	NL	D	G	0	no	yes	Vispipeline
GDA [12]	NL	D	G	1	no	yes	DR Toolbox
GPLVM [111]	NL	D	G	1	no	no	DR Toolbox
Fast ICA [93]	LIN	S	G	2	yes	yes	scikit-learn
IDMAP [142]	NL	S	L	3	no	yes	Vispipeline
Isomap [203]	NL	S	L	1	yes	yes	scikit-learn
Landmark Isomap [33]	NL	S	L	1	no	no	Vispipeline
LAMP [95]	NL	S	L	3	yes	no	Vispipeline
Laplacian Eigenmaps [17]	NL	D	L	0	no	no	scikit-learn
LLC [201]	NL	S	L	3	no	yes	DR Toolbox
LLE [175]	NL	S	L	3	yes	no	scikit-learn
Hessian LLE [46]	NL	S	L	3	yes	no	scikit-learn
Modified LLE [229]	NL	S	L	3	yes	no	scikit-learn
LMNN [215]	LIN	S	L	3	no	yes	DR Toolbox
LPP [83]	LIN	S	G	1	yes	yes	Tapkee
LSP [154]	NL	S	L	4	no	yes	Vispipeline
LTSA [230]	NL	S	L	3	yes	no	scikit-learn
Linear LTSA [228]	LIN	S	L	1	no	no	Tapkee
Manifold Charting [25]	NL	S	L	2	no	yes	DR Toolbox
MCML [72]	NL	S	L	0	no	no	DR Toolbox
MDS [206]	NL	D	G	2	no	no	scikit-learn
Landmark MDS [189]	NL	D	G	1	no	no	Tapkee
Nonmetric MDS [109]	NL	S	G	2	no	no	scikit-learn
Landmark MVU [217]	NL	S	G	2	no	no	DR Toolbox
NMF [113]	LIN	S	G	4	yes	no	scikit-learn
PBC [153]	NL	S	L	4	no	yes	Vispipeline
PCA [97]	LIN	S	G	0	yes	yes	scikit-learn
Incremental PCA [174]	LIN	S	G	0	yes	no	scikit-learn
Kernel PCA (polynomial) [182]	NL	S	G	1	yes	no	scikit-learn
Kernel PCA (RBF) [182]	NL	S	G	1	yes	no	scikit-learn
Kernel PCA (Sigmoid) [182]	NL	S	G	1	yes	no	scikit-learn
Probabilistic PCA [205]	LIN	S	G	1	yes	yes	DR Toolbox
Sparse PCA [231]	LIN	S	G	3	yes	yes	scikit-learn
PLSP [155]	NL	S	G	0	no	yes	Vispipeline
Random Projection (Gaussian) [43]	NL	S	G	0	yes	no	scikit-learn
Random Projection (Sparse) [43]	NL	S	G	0	yes	no	scikit-learn
Rapid Sammon [156]	NL	S	G	0	yes	no	Vispipeline
t-SNE [127]	NL	D	L	3	no	no	Multicore TSNE
SPE [2]	NL	S	G	2	no	no	Tapkee
Truncated SVD [80]	LIN	S	G	1	yes	no	scikit-learn
UMAP [138]	NL	D	L	3	yes	no	umap-learn
dt-SNE [166]	NL	D	L	4	no	no	dt-SNE repository
NNproj [52]	NL	S	L	network size	yes	yes	NNProj code

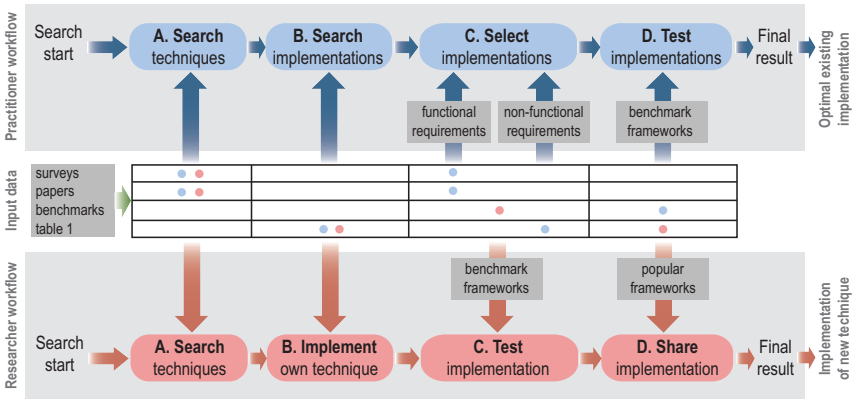


Figure 4.1: Workflows for selecting DR implementations for practitioners (top, see Section 4.3.1) and researchers (bottom, see Section 4.3.2). Colored dots indicate which input data (surveys, papers, benchmarks, Table 7) are used by the two roles in each step.

B. Search implementations. *Where do I find implementations?* Table 7 provides our survey of available implementations, with frameworks providing these in Table 8. Related to the search process, we observed several points. Since many data science projects use Python, practitioners would likely favor Python-based DR implementations. We found out that scikit-learn [1] provides high-quality implementations of many well-known DR techniques. However, other techniques come in different languages (Tapkee [117]: C++; Van Der Maaten's DR Toolbox [129]: MATLAB; and Vispipeline [96]: Java). These require manual Python wrapping, which is not easy for the average user, and may hamper adoption of less known DR techniques, which are not available anywhere else.

Table 8: DR frameworks used in the study.

Framework name	Available at	Programming Language(s)	Documentation Quality
DR Toolbox	lvdmaaten.github.io/drtoolbox	MATLAB	−−
Multicore TSNE	github.com/DmitryUlyanov/Multicore-TSNE	Python and C++	−−
scikit-learn	scikit-learn.org	Python	++
Tapkee	tapkee.lisitsyn.me	C++	−−
umap-learn	github.com/lmcinnes/umap	Python	++
Vispipeline	vicg.icmc.usp.br/vicg/tool/1/projection-explorer-pex	Java	−−
Keras	keras.io	Python	++
NNP repository	github.com/mespadoto/dlmp	Python	+

C. Select implementations. *How to do this selection?* This involves considering both functional and non-functional requirements. For the former, one can easily screen DR techniques based on their properties listed in recent surveys [56, 148]. For the latter, we point to our own survey in Tabs. 7 and 8. Based on both requirement types, a ranking is made and a subset of candidates are selected. We observe several points concerning non-functional requirements. Regarding *documentation*, outside of scikit-learn and UMAP, which are well documented, most libraries we found had little to no documentation at all (Table 8), making adoption hard. In some cases, reading the source code is the only option, like in the case of many techniques found in Van Der Maaten's DR Toolbox. Even in the cases where the library is well documented, we often found not

enough details on the role of each parameter in the final quality of the projection, and even less on the interaction between parameters. Separately, the *API design* promoted by DR libraries can vary enormously. Each library has its own conventions on data format and parameters, which makes the problem of interfacing hard. Take, for example, the examples below, which run t-SNE [127] on some dataset X using scikit-learn and Tapkee (Listings 1 and 2). We believe the two-step API of scikit-learn (create object with parameters, call fit_transform()) to be much simpler to understand for the average user than Tapkee's method of chaining with globally-namespaced, non-specific keywords.

Listing 1: t-SNE example with scikit-learn

```
from sklearn.manifold import TSNE
tsne = TSNE(perplexity=30)
output = tsne.fit_transform(X)
```

Listing 2: t-SNE example with Tapkee

```
using tapkee;
TapkeeOutput tsne = initialize()
  .withParameters((
    method=tDistributedStochasticNeighborEmbedding,
    target_dimension=2,
    sne_perplexity=30))
  .embedUsing(X);
auto output = tsne.embedding.transpose();
```

D. Test implementations. *How to test the selected implementations?* Surveys and code inspection cannot tell everything about a specific DR implementation. One needs to actually test an implementation on given datasets and against a set of metrics. For this, *benchmarks* are needed. To our knowledge, only three such benchmarks exist in DR landscape [56, 139, 209]. Yet, such benchmarks are never complete, so they need to be extended with additional DR implementations, datasets, and metrics. We discuss how such benchmarks can be designed for extensibility next in Section 4.4.

4.3.2 *Researcher workflow*

A. Search techniques. This step is largely similar to step A for practitioners (Section 4.3.1). The focus is though different for researchers, who are mainly interested in finding *functional* limitations of existing DR techniques that they want to improve upon, rather than the non-functional ones that are more relevant to practitioners.

B. Implement own technique. This step can proceed, for the most part, independently from existing DR implementations. However, some researchers may choose to follow coding standards and API conventions of existing (successful) DR implementations to already maximize exposure at this stage.

C. Test implementation. This step typically uses the same benchmarks as in step D for practitioners. An important part of this step is *presenting* the test results. While less important for practitioners, researchers need compelling ways to show that their (novel) techniques perform well *vs* many other techniques on many metrics to convince their

peer researchers. For this, metric *visualizations* are used, the most prominent being bar and boxplot charts [128], tables [148], and space-filling charts [56]. A challenging aspect here is that the space to visualize is at least four-dimensional (datasets, metrics, DR techniques, parameter settings). Ideally, creating such visualizations, and adding new visualizations, should be supported by the benchmarks.

D. Share implementation. Once a DR implementation has been successfully tested, its further impact critically depends on how easily it is *shared* with practitioners. Doing this follows two approaches. *Standalone code* is the easiest way – the researcher just makes her DR implementation available via a website or software repository. Examples hereof are dt-SNE, NNproj, and DR Toolbox. One issue with this approach is that integrating DR code having non-standard APIs with other components of a data science pipeline can be hard. Also, standalone code is less visible to practitioners than code shared via well-known frameworks (discussed next). Finally, adaptive maintenance is less often done on standalone code, which can easily break it upon the evolution of third-party components it uses. An example hereof is dt-SNE, which depends on the unmaintained Theano [21] library. However, good examples of standalone code sharing exist, such as t-SNE archive and UMAP (Table 8), discussed below. *Framework integration*, the second approach, adds the DR implementation to a mainstream data science or similar framework. Examples hereof are scikit-learn, Tapkee, MATLAB, and Vispipeline. This takes considerably more effort than sharing standalone code, as the code to integrate must comply with framework APIs and documentation constraints, but favors (far) larger exposure. Framework integration can be hard for DR techniques which need more than data input-output communication with the framework. Examples hereof are projections which advertise landmarks interactively placed by the user, such as LAMP [95] or for techniques which use GPU acceleration, *e.g.* TFJS-t-SNE [160]. To integrate these, a framework should provide APIs for interaction, respectively for GPU computing.

Standalone code sharing has limitations, but can still be very successful. Two good examples are UMAP [138] and t-SNE [127]. UMAP's author did an excellent job of providing a Python library which is API-compatible with scikit-learn, easy to install and use, and well documented. We believe the availability of the UMAP library via the standard Python package manager and its scikit-learn compatibility are key factors (apart from UMAP's quality) of its growing popularity. In contrast, the t-SNE archive (maintained by Van der Maaten) chose to provide a 'portal' for implementations in several languages (C++, Python, JavaScript, CUDA, R, Java, MATLAB), thereby simplifying integration with third-party code in all these languages.

4.4 ARCHITECTING AN EVALUATION BENCHMARK

Having a *benchmark* to quantitatively evaluate DR algorithms is essential for both the practitioner and researcher workflows (see Section 4.3). As mentioned there, not many such benchmarks exist. Importantly, by a benchmark, we do not understand here just a 'passive' collection of high-dimensional datasets to battle-test DR techniques, but rather a *runnable* software system that lets one select datasets, DR technique implementations (and their parameter values), metrics, execute them, and visually inspect the results in an easy and highly automated way. The only two benchmarks that approach this definition are [56] (for static DR techniques) and [209] (for dynamic DR techniques). However, these benchmarks have their own limitations. Extending them involves, at points, man-

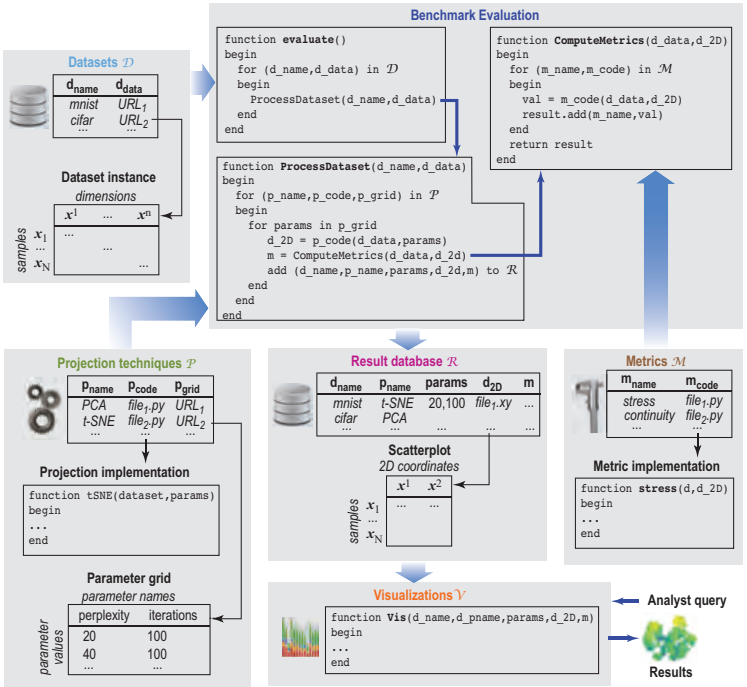


Figure 4.2: Proposed benchmark architecture and its dataflow execution workflow (Section 4.4).

ually reading and reverse-engineering their code, which is hard. Creating an even better (broader, easier to use) benchmark is a high-effort task involving many decisions.

We aim to support the interested users in either the extension task or the design-from-scratch task by proposing a generic architectural template for such a benchmark (see Fig. 4.2. We created this architecture by studying the two benchmarks [56, 209], including the implementation [55] of the former, and next unifying their design and implementation, aiming to generalize and simplify. We believe that our proposal meets well the genericity and extensibility requirements, as detailed next.

Overall design: The benchmark follows a dataflow execution model (see *execute* function in Fig. 4.2). Datasets d from a database $\mathcal{D}$ are projected in turn by several DR technique implementations p from a DR technique collection $\mathcal{P}$, using several parameter values $params$, yielding corresponding 2D scatterplots $d_{2D} = p(d, params)$. For each such scatterplot, a set of projection quality metrics $m(d, d_{2D})$ is computed. The results d_{2D} and m are stored in a result database $\mathcal{R}$, implemented using the Python "pickle" binary format files for efficiency. These results can be next visually explored by visualizations selected by the analyst from a given set $\mathcal{V}$. We next detail each of the main components outlined above.

Dataset collection $\mathcal{D}$: This is the set of datasets to be considered in the evaluation, stored as a name-value dictionary (d_{name}, d_{data}). The values d_{data} are URLs pointing to actual files that store the data samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$, each having n dimensions $\mathbf{x}^1, \ldots, \mathbf{x}^n$, in a table format, using either CSV following the 'tidy data' [218] standard or binary

NumPy [212].

Projection techniques $\mathcal{P}$: This is the set of DR technique implementations to be evaluated, as well as their parameters to be used during evaluation, stored as a set of tuples $(p_{name}, p_{code}, p_{params}$. Here, p_{code} points to the Python implementation of a DR technique, which is a function that expects a dataset $dataset$ and parameter-set $params$, and returns the computed 2D scatterplot d_{2D}. p_{params} stores a so-called *parameter grid*, *i.e.* a table having as many columns as parameters p expects, and one row per parameter-set to be used during the evaluation. For instance, if we want to evaluate a t-SNE implementation of p that expects two parameters *perplexity* and number of *iterations* (see [127] for details on these parameters), which range in $perplexity \in \{20, 40\}$ and $iterations \in \{100, 150, 200\}$, we provide a parameter-grid p_{grid} table equal to the Cartesian product $\{20, 40\} \times \{100, 150, 200\}$. This design allows flexibly evaluating DR techniques having different numbers and types of parameters over user-supplied parameter grids.

Metrics $\mathcal{M}$: This is the set of quality metrics to be used to assess the benchmarked projections, stored as a dictionary $(m_{name}, m_{code}$. Here, m_{code} points to the Python implementation of a metric, which is a function that expects a dataset $d \in \mathcal{D}$ and its 2D projection d_{2D} computed by one of the techniques $p \in \mathcal{P}$, and returns a metric value (typically a scalar). As for projections, this design allows easily incorporating any of the projection quality metrics known in the literature.

Visualizations $\mathcal{V}$: This is the set of visualization tools offered to the analyst to explore an evaluated benchmark. Each visualization *Vis* is a Python function receiving a tuple $(d_{name}, p_{name}, params, d_{2D}, m)$. If a parameter is set to the predefined value *each*, then the visualization will generate separate small-multiples for each of the values of that parameter in $\mathcal{R}$. If a parameter is set to the predefined value *aggregate*, then the visualization will generate a single plot for all values of that parameter. These two options are conceptually analogous to the SQL operations *SELECT **, respectively *SELECT SUM*. This allows one to easily specify visualizations having different levels of data aggregation. For example, if we want to display quality metrics, setting $d_{name} = each, p_{name} = each$ creates one separate metric plot for each different pair of dataset and DR technique in $\mathcal{R}$. Setting $d_{name} = each, p_{name} = aggregate$ creates one metric plot that shows, for each dataset, the aggregate (*e.g.*, average, depending on the actual *Vis* implementation) values of metrics over all DR techniques. Finally, setting a parameter to a specific value, *e.g.* $p_{name} = t\text{-}SNE$, creates a visualization only for the respective DR technique entries present in $\mathcal{R}$. The dictionary keys $d_{name}, p_{name}, m_{name}$ show now their purpose: They are used both for the user to select specific entries in $\mathcal{R}$ to visualize and to create labels in the generated visualizations. This design allows specifying a quite large range of visualizations, see the examples in [56, 209].

In terms of implementation, actual visualizations can be coded as Python scripts calling Matplotlib [92] (as in [56]). A more interactive and flexible development workflow can be achieved by using Jupyter notebooks that allow for independent and interactive execution of their code cells and rich presentation of their output (visualizations, narrative text, mathematical equations, tables). Execution automation is supported by Papermill [151] which allows the parameterization, instantiation, and execution of Jupyter notebooks. For example, to generate for each dataset d a separate video showing a small-multiple display of all its time-dependent projections $p(d)|d \in \mathcal{D}$, one can write a template video generation notebook and use Papermill to instantiate a new notebook

for each dataset d.

Extensibility and genericity: The above architecture is easily extensible: Adding new datasets, metrics, DR techniques, or parameter grids implies simply adding entries to the respective dictionaries. Dictionaries can be implemented either in Python or, even simpler, as folders having filenames as keys and the respective file contents as actual values. For large benchmarks, implementations using relational databases (*e.g.*, SQL) could also be done relatively easily following the same template architecture. The architecture is also generic, since the formats of datasets, respectively signatures of functions implementing DR techniques, metrics, and visualizations, can accommodate most, if not all, concrete instantiations thereof that we know of from the DR literature and practice. For instance, all DR technique implementations in Table 7 fit this architecture. Parallelization can also be easily incorporated, *e.g.*, by simple running of multiple processes at the level of the for-loops in *execute* (a design used in [56]). Finally, supporting time-dependent DR techniques implies only a small change to the architecture, namely having d_{data} point to a *sequence* of tables rather than a single one (as done in [209]).

4.5 DISCUSSION

Summarizing, we see a number of open challenges to the selection and sharing of DR algorithms that, jointly, create a gap between the state-of-the art work in DR literature and practical realities in the field, as follows.

Implementation: For *selection* (adoption), we note the lack of analysis of DR *implementations* with respect to non-functional requirements such as programming language, documentation quality, and ease of use. These make practitioners stay away from techniques whose implementations score poorly along these requirements and, conversely, favor techniques that have good-scoring implementations. For sharing, we see that there is no single framework that provides implementations of most DR techniques known in the literature – the closest to this is scikit-learn which implements roughly half of the DR techniques for which we found a mainstream implementation. For *sharing*, we do not see yet a momentum for researchers to develop DR algorithms within mainstream data science frameworks – the dominant sharing form is still standalone code.

Benchmarks – Datasets: Selecting a representative collection of datasets to gauge DR techniques is hard. Typically, papers, surveys, and benchmarks use datasets that are known in DR literature (for historical reasons) or in a given application domain. However, gauging the quality of a DR technique *at large* should use datasets that ideally represent well any problem. Espadoto *et al.* [56] do a first attempt in this direction by characterizing datasets by traits (*e.g.*, dimensionality, intrinsic dimensionality, sparsity, provenance) and create a benchmark by sampling these dimensions. Vernier *et al.* [209] use the same idea and aim to also cover dataset dynamics. However, both these surveys admit to only very sparsely sample the space of all possible datasets. Coming up with a good such sampling is an open, and important problem, in DR practice.

Benchmarks – Techniques: To our knowledge, most DR surveys and benchmarks focus on techniques that handle quantitative data, and static projections (except [209]).

Adding DR techniques that handle other attribute types such as categorical is another open direction towards creating comprehensive benchmarks.

Benchmarks – Metrics: DR literature knows tens of different quality metrics [22]. However, existing benchmarks implement only very few – the most being, to our knowledge, the 6 metrics in [56]. A benchmark with a wide set of readily-implemented metrics would be of high value to both practitioners and researchers.

Benchmarks – Extensibility: Some DR benchmarks [56] provide code that allows the experiments to be reproduced and allow for some extensibility, in terms of adding new projection techniques, metrics and datasets. In terms of storing results and creating visualizations, users would benefit from a more structured approach, with data saved in portable formats, and with some form of integration with popular data visualization and exploration tools, such as Tableau [31].

4.6 CONCLUSIONS

We presented an overview of the challenges that exist in the process of selecting and sharing DR techniques from the point of view of different audiences, practitioners and researchers, where we described the typical workflows used in their processes. We listed and ranked the most common sources of information about DR techniques, namely, papers, surveys, benchmarks, and frameworks, and compiled a list with popular frameworks and techniques. We described the architecture used in two recent benchmarks and showed how these can be extended to consider more datasets, techniques, and metrics.

We believe the visualization community would benefit from a more integrated, well-documented benchmark framework, where new techniques, datasets, and metrics could be easily added by the user with minimal programming, and with the capability of integration with existing visualization tools.

In Chapters 5, 6 and 7 we develop the idea of using Neural Networks to create direct and inverse projections. We present techniques and results for several experiments that show how Neural Networks can be used to create projections that have high quality and are very scalable. In the work presented in those chapters, we (re)use the evaluation framework presented in this chapter, thereby providing additional confirmation to its practical added value.

5 DEEP LEARNING PROJECTIONS

5.1 INTRODUCTION

In Chapter 3, we detailed the importance and usefulness of multidimensional projection techniques to the analysis of high-dimensional data, and also explored and explained in detail the differences between several tens of projection techniques. In particular, following that evaluation, we saw that t-SNE [127] is one of the top-ranking techniques, which is in line with its popularity. Yet, t-SNE comes with some downsides: It is very slow for large data sets (thousands of observations or more), due to its quadratic nature; its parameters can be tricky to get right, which can lead to unpredictable results [214]; and it lacks the capability of projecting out-of-sample data, which is useful when comparing several (time dependent) datasets [124, 148, 166]. Additionally, Chapter 3 shows that no single projection technique is perfect – different techniques have different pro's and con's. This makes it interesting to consider ways in which one can reduce the limitations of any such technique, while keeping its advantages.

Work has been done to address the performance issue, such as tree-accelerated SNE [126], H-SNE[158], A-SNE[159], and UMAP[138], which is a completely different algorithm but with the stated goal of having t-SNE quality at a higher speed. However, in general, there is no technique in the t-SNE class that jointly addresses scalability, stability, and out-of-sample handling. Moreover, t-SNE refinements are algorithmically not simple to understand and/or implement, which may limit their attractiveness. Such limitations are, to a large extent, shared by many other projection techniques [148], so, our discussion next should be seen in this larger context rather than focusing on t-SNE class methods only. A way to handle these limitations *jointly* and *independently* on the projection technique of choice is of considerable interest.

To address the above goal, we propose a learning-based approach to dimensionality reduction: We take any projection technique (deemed suitable for an application at hand), run it on a small subset of the available data, train a deep neural network to learn the mapping from high to low dimensional space produced by the respective projection, and use the trained network to project the entire dataset or similar datasets. We call this technique Neural Network Projection, or NNP. This technique has the following contributions:

Quality (C1): it provides similar results to the learned projection, as measured by several well-known metrics in dimensionality reduction literature; briefly put, such metrics quantify how close the 2D projection reflects the structure (inter-point distances and nearest neighbors) of the high dimensional dataset;

Scalability (C2): it computes projections in linear time in the number of dimensions *and* observations. Practically, we project datasets of up to a million observations and hundreds of dimensions in a few seconds using commodity hardware, no matter which

This chapter is based on publication [52]

projection technique we are emulating;

Ease of use (C3): it works without the need to set any complex parameters, except for the number of training epochs, which is minimized by the use of early stopping (see Section 5.4.1). NNP is implemented using only open-source infrastructure, so it is easily replicable;

Genericity (C4): it can handle any kind of high-dimensional data that can be represented as high-dimensional vectors, and can mimic the behavior of different types of projection techniques;

Stability and out-of-sample support (C5): NNP allows one to project new observations for a learned projection without recomputing it, as is needed with standard t-SNE and any other non-parametric methods;

This chapter is structured as follows. Section 5.2 discusses related work on multidimensional projections. Section 5.3 details our method. Section 5.4 presents our results that support our contributions outlined above. Section 5.5 discusses our proposal. Section 5.6 concludes the chapter.

5.2 RELATED WORK

Dimensionality reduction: Over the years tens of Dimensionality Reduction (DR) methods have been developed. These propose quite different trade-offs between the desirable features listed in Section 5.1. Please refer to Chapter 3 for a thorough discussion of dimensionality reduction techniques.

Deep learning: Neural network approaches have been proposed for DR, such as autoencoders [88, 101], which aim to generate a compressed, low-dimensional representation on their bottleneck layers by training the network to reproduce its inputs on its outputs. Typically, autoencoders produce results comparable to PCA on the quality criterion (low C1). Also, for different types of datasets, one typically needs to design different autoencoder architectures, which is time-consuming (low C3). Yet, autoencoders are easily parallelizable (high C2), predictable, and have out-of-sample capability (C5).

The ReNDA algorithm [15] is a very recent neural-based approach that uses two networks, improving on earlier work from the same authors. One network is used to implement a nonlinear generalization of Fisher's Linear Discriminant Analysis, using a method called GerDA; the other network is an Autoencoder used as a regularizer. According to the results of the original paper, the method scores well on predictability and has out-of-sample capability (C5). However, it requires labeled data, which none of the other algorithms discussed in this study do. Also, it does not scale well for large datasets (low C2).

Parametric t-SNE (pt-SNE) [124] was proposed to address some of the limitations outlined in Section 5.1. pt-SNE uses a deep learning architecture consisting of Restricted Boltzmann Machines (RBMs) [86] to pre-train a neural network to reduce dimensionality, followed by a fine-tuning stage that refines this network to minimize a cost function based on t-SNE's Kullback-Leibler (KL) divergence. The key advantage of pt-SNE is its parametric nature (mapping the entire nD input space to the lower-dimensional qD space), which allows out-of-sample behavior by construction. Only few DR methods in existence are parametric and thus have this ability (*e.g.*, PCA [97], NCA [74],

autoencoders [88]). From these methods, PCA and NCA do not work well when the intrinsic dimensionality of the input data is higher than the one of the output space, due to their linear nature. More interestingly, a detailed discussion on why pt-SNE is superior to autoencoders in terms of quality (criterion C1) is provided: pt-SNE aims to capture the *local* data structure in nD, using Gaussian distributions, and to transfer this structure to the low-dimensional qD space using the KL divergence cost. In contrast, the cost function of autoencoders aims to maximize data *variance* in qD. This does not create well-separated clusters in qD, even when these exist in nD, as this would decrease variance and thus increase the reconstruction error. Similar to pt-SNE, our method is also parametric (thus satisfying C5), uses a deep learning approach, and has the same advantages *vs* autoencoders. In contrast to pt-SNE, however, we have a much simpler neural network architecture, and thus a simpler training process, and a single hyperparameter (number of training epochs or, alternatively, training loss); we use *supervised* learning (with a given 2D scatterplot produced by a user-chosen projection method as ground truth), and thus have a completely different cost function (distance to 2D ground-truth projection rather than KL divergence); and, finally, our method can learn *any* projection technique, not just t-SNE.

5.3 METHOD

Our proposal is very simple: Consider a data *universe* $\mathcal{D}$, *i.e.*, the union of all data sets created by a given application area, *e.g.* all fashion images, all handwritten digit images, or all astronomical images related to a certain type of measurement. If we admit that there exists some specific structure of the data in such a universe, *i.e.*, the data samples are not uniformly distributed along all dimensions, then a *good* projection should capture well this data structure (which is *e.g.* reflected in terms of segregating different data clusters in the visual space). We hypothesize that the way in which a given projection technique P captures this data structure can be *learned* by using a limited number of small training datasets $D \subset \mathcal{D}$ and their respective projections $P(D) \subset \mathbb{R}^2$.

Our proposal follows precisely this: Let D_s be a randomly-selected subset of one or several datasets $D \subset \mathcal{D}$, and let $P(D_s)$ be the corresponding projection of D_s. Let P_{nn} be a neural network trained on D_s aiming to mimic the behavior of $P(D_s)$. Let $D_p = D \setminus D_s$ be the remaining data in D to be projected by P_{nn}.

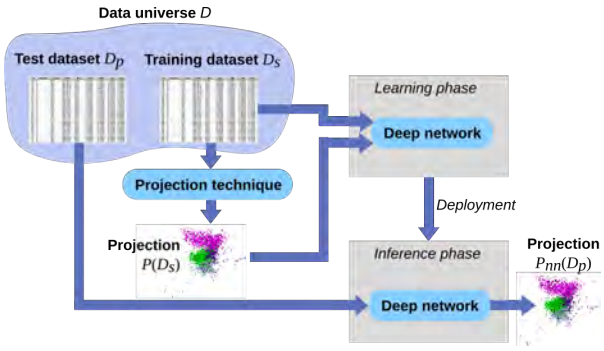


Figure 5.1: Pipeline for learning projections (see Section 5.3).

Figure 5.1 presents our idea, which consists of three main steps – creating the training projection, training, and inference. To create the training projection, we project D_s by using any user-chosen projection technique P. We next use the projected subset $P(D_s)$ alongside the original high-dimensional D_s to train a feed-forward, fully-connected neural network P_{nn} to learn how to project high-dimensional data. Once P_{nn} is trained, we use it to project the remaining points D_p of D. By extension, we use P_{nn} to also project different datasets from the same universe $\mathcal{D}$ as D.

After empirical testing, varying the number of layers and the number of units in each layer, we defined the architecture for P_{nn} as having three fully-connected hidden layers, with 256, 512, and 256 units respectively, using ReLU activation functions, followed by a 2-element layer which uses the sigmoid activation function to encode the 2D projection, scaled to the interval $[0, 1]^2$ for implementation simplicity (Fig. 5.2). The number of units in the input layer matches the dimensionality of the input data (n in Equation (2.1)). We stress that this particular network architecture has nothing special about it. Other similar architectures may work as well. The central goal in this chapter is to propose a novel way to achieve dimensionality reduction by learning projections. In Chapter 6 we present a detailed study of different architectures and hyperparameters for this technique.

We initialize weights with the He uniform variance scaling initializer [82], and bias elements using a constant value of 0.0001, which showed good results during testing. We use the Adam [102] optimizer, a variant of the well-known Stochastic Gradient Descent, to train P_{nn} for at least 10 epochs and up to 200 epochs on an "early stopping" setup. That is, training automatically stops on convergence, defined as the epoch where the validation loss stops decreasing. In practice, no more than 60 epochs are needed to achieve convergence, the average being 30 epochs (see Section 5.4.1). The cost function used is mean squared error (MSE, Equation (5.1))

$$MSE = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|^2 \tag{5.1}$$

where $\mathbf{y}_i$ are the ground truth 2D coordinates provided by the training projection and $\hat{\mathbf{y}}_i$ are the 2D coordinates predicted by the network, respectively. MSE showed higher convergence speed during testing than other common cost functions such as mean absolute error and log hyperbolic cosine (logcosh).

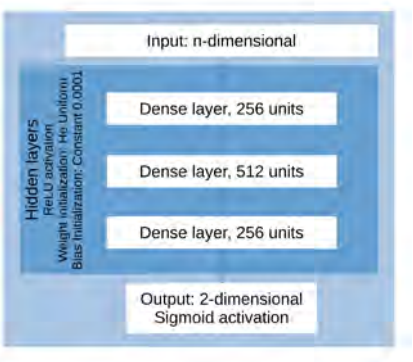


Figure 5.2: Feed-forward network architecture used.

We *test* P_{nn} by comparing the projections it delivers on D_p (unseen data during training) with the ground truth $P(D_p)$ obtained by running the projection P we desire to

mimic on D_p. For this, we use two classes of metrics, as follows. *General metrics* capture desirable properties of a projection $P(D)$ in a technique-agnostic manner, that is, without considering the specific objective (cost) function that P tries to optimize. Such metrics are often used in projection literature to compare different techniques that do not share implementation similarities [56, 148]. *Technique-specific metrics* consider the cost functions used by specific techniques, and thus allow comparing these techniques to our method, *i.e.*, show how well our method manages to learn the 'style' of a projection by exposing how well it optimizes the underlying cost function. The definition of all metrics can be found in 2.2.

5.4 RESULTS

We next show how our proposal covers the requirements listed in Section 5.1. For this, we structure our evaluation into several tasks. We compare our results with those produced by several well-known projection techniques (t-SNE, UMAP, PCA, Isomap, MDS, LAMP, LSP, pt-SNE, and LLE). We use a range of publicly available real-world benchmark datasets (See Section 2.3 for details) that have many observations and dimensions, exhibit a non-trivial data structure, and come from different application domains, as follows (Fig. 5.3 shows samples from these datasets):

Figure 5.3: Illustrative examples from the used datasets.

For each dataset, the split between training and test sets varies for each experiment and is explained in detail next in each task-specific section.

5.4.1 *Training effort*

It is important to assess what our method needs (training-data-wise and training-effort-wise) to reach the quality of the training projection, or close to that. Figure 5.4 shows t-SNE, UMAP, MDS, and LAMP projections of subsets of the MNIST dataset with two and ten classes, respectively, alongside our method's results. We used training sets D_s of varying sizes, all randomly and independently sampled from the MNIST dataset. We included the two-class selection (digits 0 and 1) since we know that images for these digits are quite different. Hence, the obtained projections should clearly visually separate samples from these two classes. For the two-class case, we see that our method yields practically the same results as the ground truth methods (t-SNE, UMAP, LAMP, and MDS, respectively), already when using only 1K training samples. For the 10-class case, we obtain very similar results starting from roughly 5K training samples. Figure 5.4 also shows how our learned projections get close to the neighborhood hit value (M_{nh}) of the ground-truth projection as we increase the training-set size. It is important to note that our method *cannot* formally exceed this ground-truth value, as we learn to mimic this ground truth, not surpass it. Hence, when learning from good-quality projections, we will achieve a high quality; when learning from less good projections, we will not surpass that quality. Section 5.4.2 revisits this point with additional examples.

Figure 5.5 also shows how the quality improves for a fixed training set (3K samples) as we increase the number of training epochs. As visible, we obtain projections already very close to the ground truth from roughly 25..50 epochs for the more complex 10-class MNIST dataset, and from roughly 10 epochs for the simpler 2-class MNIST dataset, respectively.

Figure 5.6 provides more insight into the training process by showing how the loss (cost) decreases *during training* as we increase the number of epochs (blue and green curves for t-SNE and UMAP, respectively), for both the MNIST and FashionMNIST datasets, when considering only 2 classes (easier problem) or all 10 classes (harder problem). The orange and red curves (for t-SNE and UMAP, respectively) shows what the loss is for the *validation* set for the network trained for a given number of epochs. Note that we did not include curves for LAMP and MDS in Fig. 5.6 to avoid overplotting, since these curves are practically identical to the ones already shown for t-SNE and UMAP. As visible, all curves converge quite quickly and similarly for all datasets, all projections. Of course, the validation loss is a bit larger than the training loss. Separately, we see that convergence is rapid for all four considered datasets. We can use these curves in practice to find how many training epochs we need for a desired maximal loss. Conversely, we can fix a preset maximal loss (in practice, 0.005) and compute the number of training iterations required for it. Table 9 shows the resulting numbers of training epochs required which we can select this way. This justifies the maximal preset of 200 training epochs (and its average of 30 epochs) mentioned in Section 5.3.

Table 9: Number of training samples *vs* number of epochs needed to obtain convergence, MNIST dataset.

Samples	t-SNE: 2 classes	t-SNE: 10 classes	UMAP: 2 classes	UMAP: 10 classes
1000	57	49	44	31
2000	30	33	21	30
3000	50	31	31	33
5000	32	21	28	23
9000	24	13	42	21

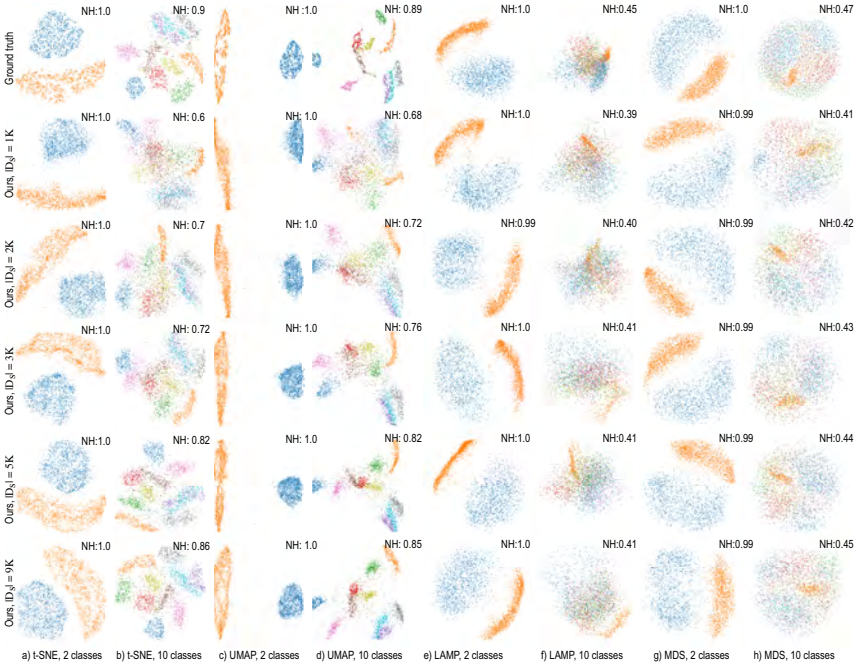


Figure 5.4: Top row: MNIST dataset, 10K sample projections of 2 and 10 classes created by t-SNE, UMAP, LAMP, and MDS. Next rows: Projections done by our method for varying training set sizes $|D_s|$. NH stands for Neighborhood Hit (M_{nh}).

5.4.2 *Capturing the structure of datasets*

In dimensionality reduction, the quality of a projection, measured by any of the metrics discussed in Section 5.3 (or any other desirable quality metric), highly depends on the kind of input data and kind of projection technique used [148]. We first assess this quality by visually comparing the results of eight different projection techniques with those created by our learning technique, trained on the respective projections. The considered techniques are t-SNE, UMAP, Isomap, PCA, LAMP, LLE, and two autoencoders using 1 layer and 3 layers respectively (AEC1 and AEC3). We included the autoencoder-based techniques as they are related to our approach as they also use deep learning. However, autoencoders work differently, since they do not use an actual 2D ground-truth to learn from, as we do. Figure 5.9 illustrates this for four datasets (MNIST, Fashion MNIST, Dogs *vs* Cats, and IMDB). Figure 5.10 adds four extra datasets to the evaluation (Spambase, Seismic, Har, and WBC). The training set sizes were 5K samples in all cases. Several observations follow.

Learning quality: We see that our method can generate projections which are visually almost identical to the ground-truth ones. However, we also see that the learned projections appear sometimes to be slightly more 'fuzzy' than the ground truth ones. This happens more for for certain (dataset, technique) combinations, see *e.g.* MNIST with t-SNE (Fig. 5.9) or Har with t-SNE and UMAP (Fig. 5.10) and far less for other combinations. The similarity of our results with ground-truth is also reflected in the neighborhood hit (M_{nh}) values: For all datasets, our method yields M_{nh} values which

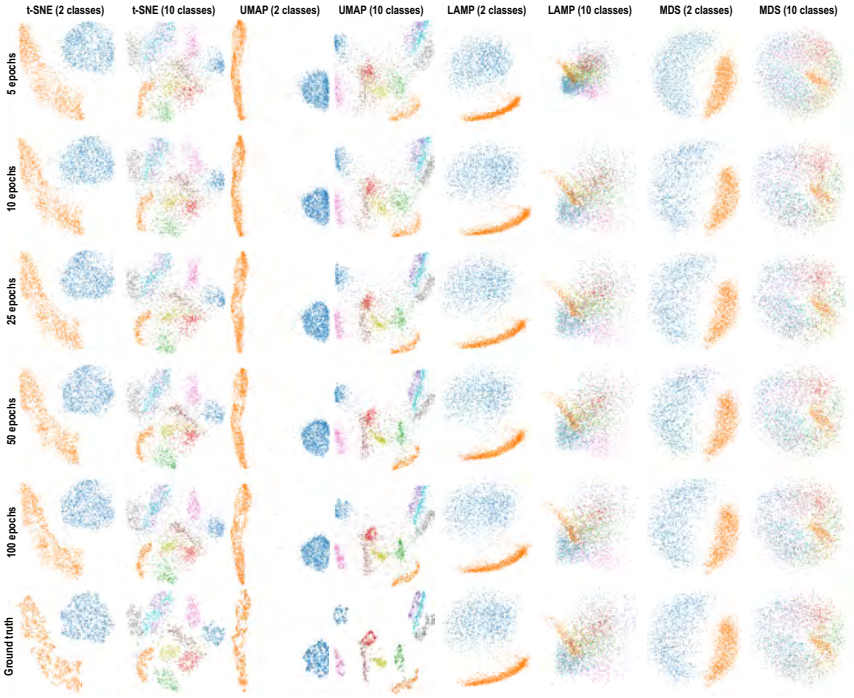


Figure 5.5: Ground truth: MNIST dataset, 3K sample projections of 2 and 10 classes created by t-SNE, UMAP, LAMP, and MDS. Rows above: Projections done by our method using varying numbers of epochs.

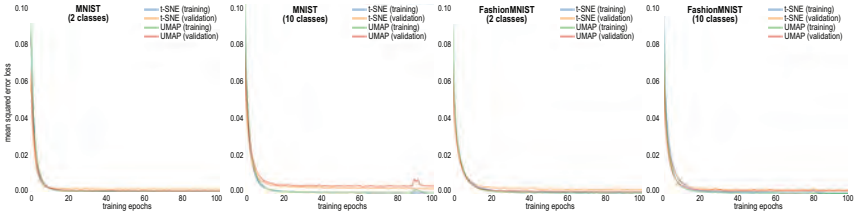


Figure 5.6: Loss as function of number of training epochs, during both training and validation, for MNIST and FashionMNIST datasets, 2-class and 10-class.

are very close to those of the ground-truth projection.

Projection quality: Figures 5.9 and 5.10 show quite clearly that different ground-truth projection techniques yield very different results in terms of visual structures for the same dataset. For example, for the Seismic dataset, PCA identifies six clearly separated compact clusters; AEC1 and AEC3 only find three clusters here; and LLE find numerous small-size clusters (Fig. 5.10). We also see that the identified clusters correlate sometimes well with class labels – for instance, all projections achieve a quite good visual separation of the cats from the dogs class in the Cats & Dogs dataset (reflected by the high M_{nh} values of all images in the respective panel in Fig. 5.9). At the other extreme, separating the two classes of IMDB is very hard for all projections. MNIST and Fashion

MNIST fall in the middle, with t-SNE and UMAP achieving good class separation, and Isomap and PCA faring worse. Hence, different projection techniques will fare very differently in uncovering structures in the data for different dataset. However, in all cases, our learned projection *mimics very closely* this ground truth. In other words, if one can find a projection technique that works well for a given dataset and task, *e.g.*, showing the correlation of data clusters with class labels, our method can learn to do the same. Conversely, if a projection technique scores poorly on a dataset, *e.g.*, cannot identify interesting structures in the data, our technique will not be able to do better.

Since we learn from the ground-truth projections by considering them as 'black boxes', we claim, although we cannot (of course) formally prove, that we can learn any projection in the same way, apart from the eight techniques demonstrated in Figures 5.9 and 5.10. This is in contrast to the only other parametric projection technique based on deep-learning that we are aware of [124], which can only learn t-SNE.

Dataset difficulty: Figure 5.7 shows additional results that illustrate how our method behaves when the 'difficulty' of the dataset to be projected increases. For this, we considered two datasets (MNIST and Fashion MNIST) which all have 10 classes of samples. The dimensions of these samples are known to be predictive of the class labels. That is, a good projection should be able to create 10 well-separated clusters of same-label samples. The difficulty of separation is also known to be higher for Fashion MNIST than MNIST. We first only consider an easy separation problem, by taking only samples of two classes of each of these datasets. In all cases, we used 5K training samples, and projected a different set of 5K samples from each of these four datasets. Figure 5.7, top two panels, show ground-truth projections computed by PCA, Isomap, MDS, and LLE for these two-class datasets. We see that all projections can separate the two classes very well (also indicated by the high M_{nh} values), and so do also our learned projections. The bottom two panels in Fig. 5.7 show the projections of the full 10-class datasets. For these more complicated datasets, we see that all the considered ground-truth techniques have considerable difficulty in separating the 10 classes well, and so do also our learned projections. Hence, we can conclude that our learned projections mimic very closely the behavior of the ground truth ones, whether this means good or poor class separation.

Table 36 in Section C.1 shows the values of the quality metrics introduced in Section 2.2 (trustworthiness M_t, continuity M_c, neighborhood hit M_{nh}, Shepard diagram correlation M_s and the technique-specific metric S_t) for the 10 projection techniques used to project the 10 datasets in Figures 5.9, 5.10, and 5.7. For each (dataset, projection) pair, we evaluate the 5 metrics both on the ground-truth projection and on our learned projection, and next study the signed difference between the former and the latter. Figure 5.8 visualizes these signed differences for all the 100 (dataset, projection) combinations. Care was taken to compute the differences in the correct direction: For the general metrics M_t, M_c, M_{nh}, and M_s, higher difference values are better, since these indicate that our method yields higher quality than the original learned methods. For the technique-specific metrics S_t, lower is better, as this indicates that our technique minimizes the underlying cost function better than the ground-truth projections. Overall, Figure 5.8 shows that the metric values of our projection are very close to the corresponding ground-truth values, the differences being maximally 4% and on average under 2%, respectively. This strengthens the insight obtained earlier by visually comparing our learned projections with the ground-truth ones that our method learns very well the characteristics of the projections it was trained to mimic. Apart from that, Fig. 5.8 shows that there is no clear correlation between high (or low) difference values and specific projection techniques or specific datasets, with the exception of the (Har, LSP) combina-

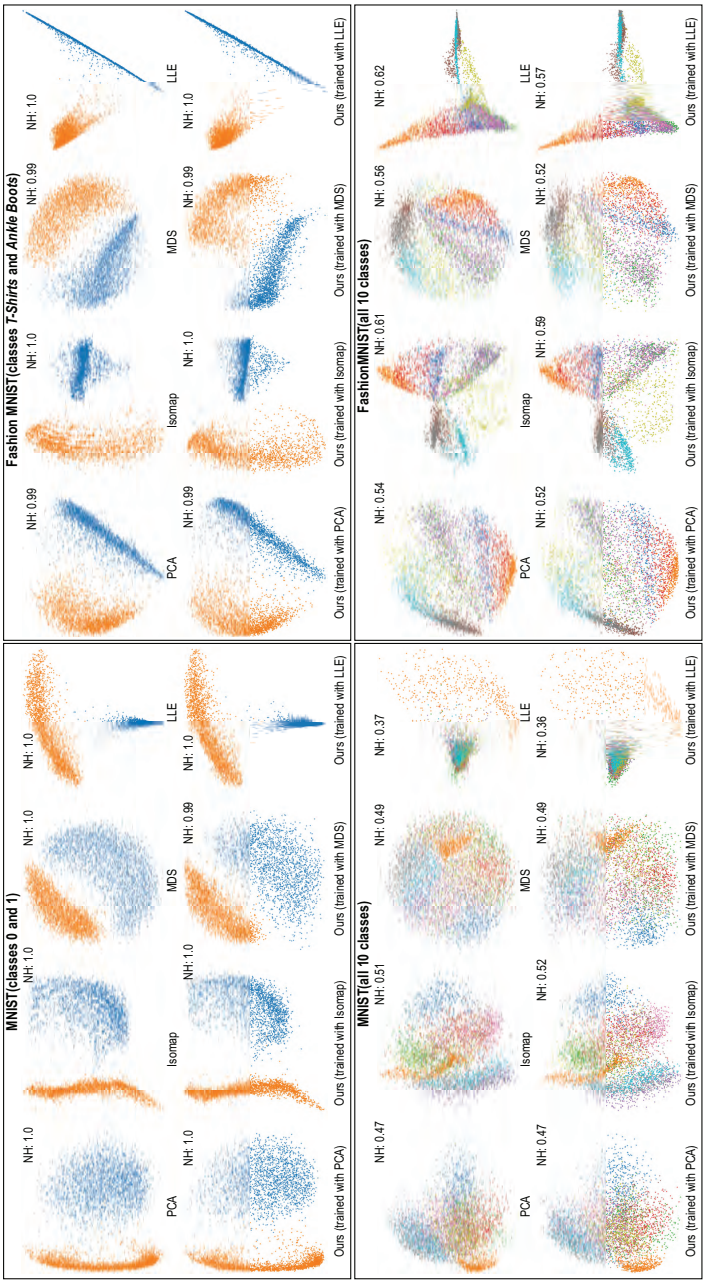


Figure 5.7: Learning different projections for four datasets (MNIST and FashionMNIST, 2 and 10 classes). In each subfigure, top row: Projections of 5K samples made with PCA, Isomap, MDS, and LLE; bottom row: projections of different 5K samples, same datasets, created by our method trained on the data and projections from the top row. *NH* stands for Neighborhood Hit (M_{nh}).

tion where our technique yields poorer M_t, M_c, and M_{nh} values than the ground-truth, but identical S_t values. In other words, our learned projections perform equally well for all techniques over all datasets.

Figure 5.8: Signed difference between quality metrics computed on ground-truth projections *vs* our projection (trained with 5K samples), for 10 techniques and 10 datasets. Green indicates cases when our technique exceeded the ground truth quality; red indicates the opposite case.

5.4.3 *Stability and out-of-sample data*

We define *stability* of a projection as the relation between the *visual* changes in $P(D)$ related to *data* changes in D. Ideally, a stable projection technique should not change $P(D)$ if D does not change at all, regardless of changes in parameters of the algorithm P; and conversely, when D changes, *e.g.* as new samples are added, then the old samples should stay in $P(D)$ as close as possible to their original locations. This way, the user can relate changes in $P(D)$ to actual data changes. For a similar reasoning applied to different infovis algorithms, *i.e.*, treemaps, see [210], or even closer to the context of dimensionality reduction, graph drawing [14]. Hence, stability and out-of-sample capabilities are closely related. Stability in the context of projecting high-dimensional data is argued for, independently, by several authors: Joia *et al.* argue for stability for preserving the user's mental map for LAMP [95] and use Procrustes analysis to align projections of different datasets to ensure consistency. Rauber *et al.* analyze deeper the trade-off stability *vs* accuracy for t-SNE and adapt the method to this end to handle time-dependent datasets [166]. Even closer to our focus, Van der Maaten [124] argues in detail about why out-of-sample

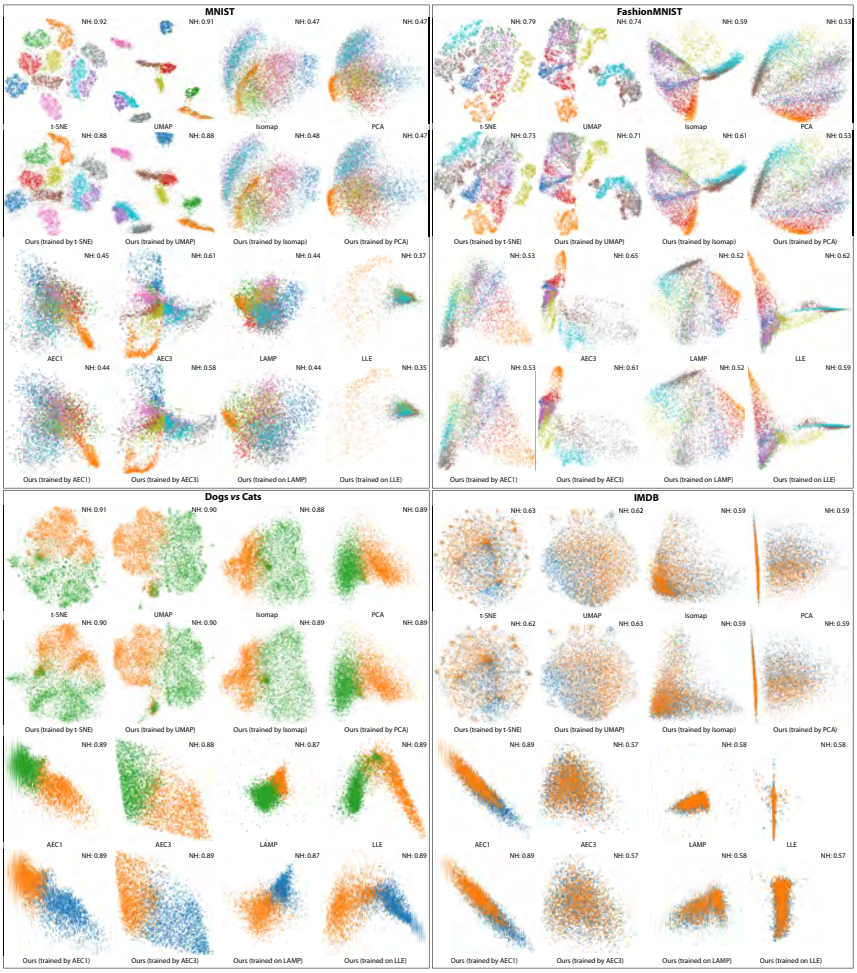


Figure 5.9: Projections (15K samples) learned from eight techniques (t-SNE, UMAP, Isomap, PCA, LAMP, 1-layer and 3-layer autoencoders (AEC1, AEC3), and LLE) for the MNIST, Fashion MNIST, Cats *vs* Dogs, and IMDB datasets. Below each projection the results of our technique are shown. See Section 5.4.2. *NH* stands for Neighborhood Hit (M_{nh}).

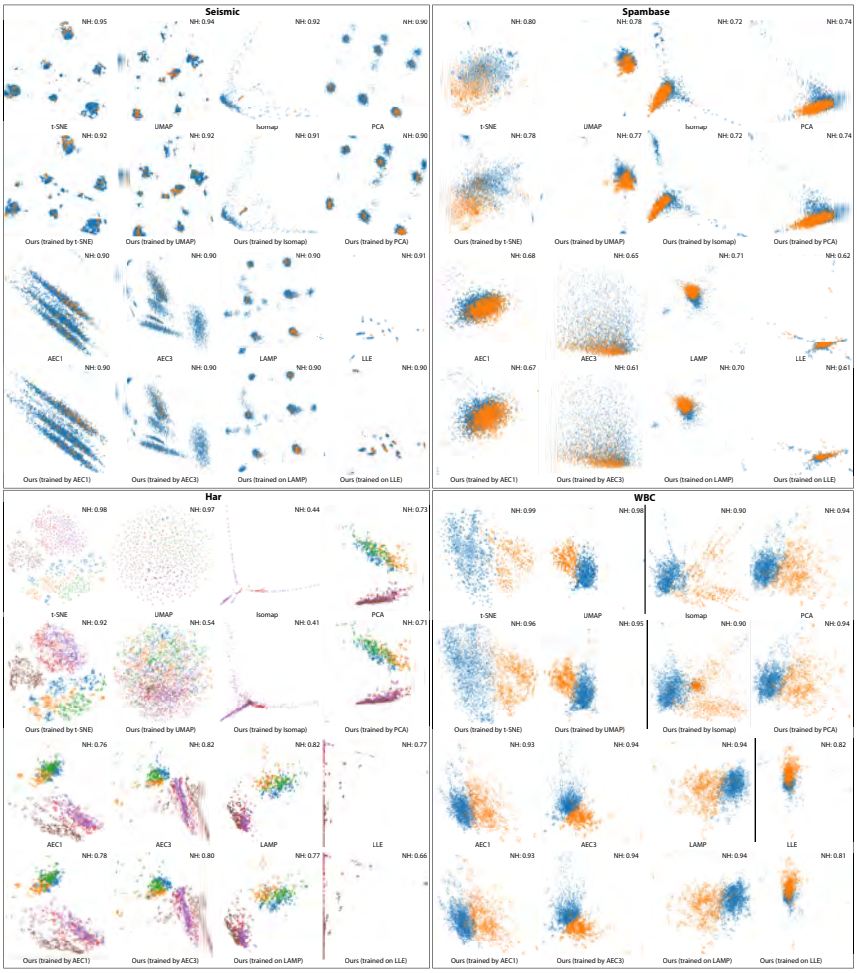


Figure 5.10: Projections (15K samples) learned from eight techniques (t-SNE, UMAP, Isomap, PCA, LAMP, 1-layer and 3-layer autoencoders (AEC1, AEC3), and LLE) for the Seismic, Spambase, Har, and WBC datasets. Below each projection the results of our technique are shown. See Section 5.4.2. NH stands for Neighborhood Hit (M_{nh}).

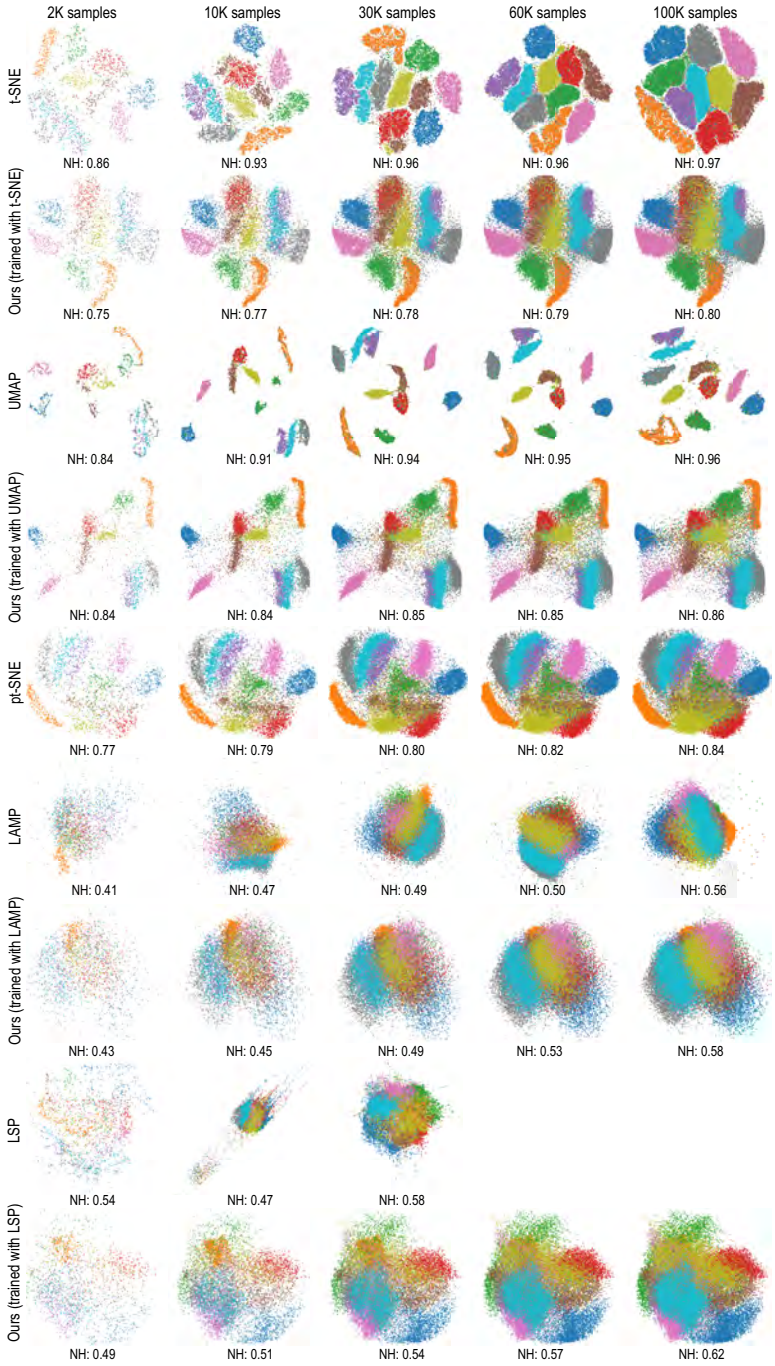


Figure 5.11: Out-of-sample capability: Projecting increasing number of samples from the MNIST dataset with different techniques. Empty spaces indicate tests that did not complete (the ground-truth projection algorithm could not handle too many samples). NH stands for Neighborhood Hit (M_{nh}).

capability for DR is essential for classification and regression, and why this is not optimally achieved by standard techniques such as autoencoders. Nonato and Aupetit argue for stability as being a key feature of DR algorithms in a comprehensive recent survey of such methods [148]. Also, one of the key criteria that UMAP was developed to satisfy its stability [138]. An additional strong argument for out-of-sample capability, and how add this to techniques such as LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering, is made by Bengio *et al.* [20].

The joint added-value of stability and out-of-sample capability can actually be explained intuitively, as follows:

Dynamic data: A dataset D may not be a *singular* item, but part of a collection $\{D_i\}$, as in the case of sampling a time-dependent some phenomenon. If projecting every single element D_i of this collection yields fundamentally different projections $P(D_i)$, even though the frames D_i don't change much with respect to each other, then the projection method P may optimally represent *the individual frames* D_i *taken separately*, but not *the entire collection* $\{D_i\}$. When analysis tasks target the collection, we need the latter, not the former, optimization [148, 166].

Usability: A projection $P(D)$ is the result of a given dataset D, *plus* any parameter settings of the projection algorithm P itself. From a practical viewpoint, users do not want to get massively different results $P(D)$ when only minute details change in either the dataset or the parameters, otherwise the algorithm P is suboptimal: Users see significantly different results for such minute changes, and next wonder whether these reflect indeed significant changes in the data D, or just noise artifacts in D or issues with parameter settings of P. An unstable DR method P does not give an answer to this, leaving the user doubting how to interpret *changes* in the visualization $P(D)$.

As outlined above, obtaining stability and out-of-sample capability is not trivial. Many projection techniques use a random initialization, which means they create quite different results for the same dataset D for different runs. Moreover, small parameter changes, *e.g.*, perplexity for t-SNE, or choice of control points for LAMP, to mention just a few, can yield large changes in $P(D)$ [214]. Dynamic t-SNE corrects such effects up to a certain level, but comes with additional complexity and significant computational costs [166].

Figure 5.11 shows projections of increasingly large, randomly-selected, point subsets of the MNIST dataset, using five projection techniques (t-SNE, UMAP, pt-SNE, LAMP, and LSP). We compare these with our method, trained on 5K samples from each of the above projections. Several observations follow. First, by scanning rows in Fig. 5.11 left-to-right, we see that in all ground-truth projections, except pt-SNE, the same-label clusters *move* in the projection as the sample count changes. This confirms that these methods are not suitable for out-of-sample applications, as users would have difficulties in maintaining the mental map of the data. In contrast, our method (trained with any of the ground-truth projections), and pt-SNE, show stable clusters that stay in the same places in the projection, and only grow denser as more samples are projected – thus, they have the desired out-of-sample capability. The price to pay for this is the increased fuzziness of our out-of-sample projections. This is especially visible when we learn from t-SNE and UMAP: For those rows, our learned projections show clusters where points having different labels (colors) mix more than in the ground-truth projections, and have lower M_{nh} values than these. However, for LAMP and LSP, our learned projections achieve similar visual quality, and sometimes even marginally higher M_{nh} values as compared to the ground truth. This is explained due to the poorer separation (quality) of the origi-

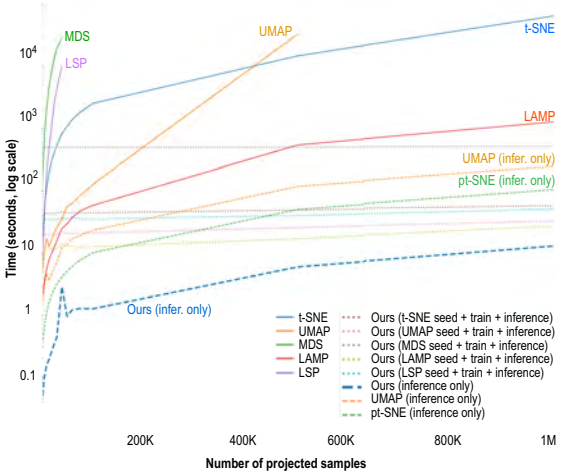


Figure 5.12: Time to project varying number of samples, MNIST data set, oversampled to 1 million observations (log time scale).

nal LAMP and LSP ground-truth, which is thus easier to learn than in the case of t-SNE and UMAP. Separately, compared to pt-SNE, the only ground-truth projection discussed here that has out-of-sample capability, we achieve only marginally lower M_{nh} values (when trained with t-SNE) and actually *higher* M_{nh} values (when trained with UMAP). Summarizing all above, we conclude that our method proposes a good trade-off between stability (and out-of-sample capability) *vs* projection accuracy.

5.4.4 *Computational scalability*

One of our main goals is to create a projection technique which scales to large datasets (C2, Section 5.1). To analyze this, Figure 5.12 shows a time comparison between t-SNE, UMAP, pt-SNE, MDS, LAMP, LSP, and our method trained to mimic these projections, for increasingly large subsets of the MNIST dataset, up to one million samples. We trained our method with (only) 5K samples, in line with training-set sizes found to be sufficient in our earlier experiments (Section 5.4.1).

We first compare the performance of our *end-to-end* method, *i.e.*, computing the training projection (called seeding in Fig. 5.12), training itself, and inference. The thin-dotted lines in Fig. 5.12 show the sum of these three times for our method. As visible, these lines are almost horizontal, which indicates that our timings are dominated by the constant seeding plus training time, and not inference.

As comparison baselines, the continuous (undotted) lines show the timings of the original ground-truth algorithms. We see that our method already runs much faster than the ground-truth algorithms, *even when considering the training cost*: For the maximum sample count (1 million), we are roughly three orders of magnitude faster than t-SNE and two orders of magnitude faster than LAMP. The LSP, MDS, and UMAP implementations we used in the test were not able to handle this large number of samples, shown by their respective curves that stop in Fig. 5.12 at roughly 20K, 30K, respectively 500K samples. From the slopes of these curves, we see that our method is several orders of magnitude faster than these algorithms for the respective sample counts.

The above comparison of our end-to-end method, including seeding, training, and inference, to the original algorithms, is a worst-case scenario: In practice, one would train only once on a given data universe $\mathcal{D}$ and project many times on the same $\mathcal{D}$. Hence, we next analyze only the inference (projection) time for our method (blue long-dash curve). This time is identical for learning any projection. We see that our method is faster than all other ground-truth methods for all sample counts. For the maximal sample count, we are about two orders of magnitude faster than LAMP, and three-and-a-half orders of magnitude faster than t-SNE. We also see that we are one order of magnitude faster than pt-SNE. This is an important result, since, as already discussed, pt-SNE is the only other parametric projection technique using deep learning that we are aware of. This, together with the quality results discussed in Figure 5.11, show that our method is a competing alternative, concerning both quality and speed, to pt-SNE.

Finally, we consider UMAP's out-of-sample capability (see Section 5.2 for details): We run UMAP on our training set, which makes it learn a function to transform the high-dimensional data to 2D. Note that this is completely different from our deep learning – it is a particular feature of UMAP's implementation, not shared by any other projection techniques we know of. Next, we let UMAP use this learned function to project the test set. In this inference-only scenario, our method (again, the blue long-dash curve) is about one-and-a-half orders of magnitude faster than UMAP (orange dashed curve).

Table 30 in Section A.2 lists all open-source software libraries used in all our experiments. Our neural network implementation leverages the GPU power by using the Keras framework. The t-SNE implementation used is a parallel version of Barnes-Hut t-SNE [125], run on all four available CPU cores for all tests. The UMAP reference implementation is not parallel, but is quite fast (compared to t-SNE) and well-optimized. Our implementation, plus all code used in this experiment, are publicly available at `github.com/mespadoto/dlmp`.

5.4.5 *Projecting unrelated data*

So far, we showed that our method can learn from a subset of a given dataset D to project unseen samples from the same D. This serves the concrete purpose of *accelerating* projections of large datasets (see *e.g.* Fig. 5.12 and related text), by training our method on the projection of a small subset thereof, followed by inference on the entire dataset. The same approach can be used when one wants to project very similar datasets, drawn from the same distribution, *i.e.*, sampling the same phenomenon.

A different question arises in this context: Can we use our method to project (infer) data which is quite different from the training data? In other words: Can we *reuse* the training done on a given type of data from some universe $\mathcal{D}$ (for which we have, for instance, sufficient training samples) to generate a network able to project data from a related, but still different, universe $\mathcal{D}'$?

To answer this question, we conducted the following experiment. We trained our method using UMAP and t-SNE projections of 2K observations from CIFAR-10 (classes *Airplane*, *Frog* and *Truck*), which can be seen as a sampling of $\mathcal{D}$, the universe of natural images of vehicle-and-animal shapes. Next, we used the trained network to project 4K observations from CIFAR-100 (classes *Trees*, *Large Carnivores* and *Vehicles 2*), which constitutes a sampling of $\mathcal{D}'$ – a universe related, but not identical to, $\mathcal{D}$. We selected these classes because they contain images that are similar perceptually between the two universes $\mathcal{D}$ and $\mathcal{D}'$, with the goal of checking the capability of generalization of our method. Note however that $\mathcal{D}$ and $\mathcal{D}'$ are quite different: While both contain images, these are of different kinds, and acquired by different procedures.

Figure 5.13 shows the obtained results. First, we show the projections obtained by directly reusing the network trained on $\mathcal{D}$. As visible, the results, shown in Fig. 5.13b, are quite far from the ground truth (classical t-SNE and UMAP projections, Fig. 5.13a). This confirms that $\mathcal{D}$ and $\mathcal{D}'$ are, indeed, quite different, so directly reusing the training from $\mathcal{D}$ to $\mathcal{D}'$ is not possible.

We next consider training the network *from scratch*, using a small number σ of 100 to 1000 samples from $\mathcal{D}'$, mimicking the situation when the user has only few available data from $\mathcal{D}'$ to train on. The results are shown in Fig. 5.13c. As visible, when we increase σ, the from-scratch training results get closer to the ground-truth (Fig. 5.13a). Next, we consider a network pre-trained on $\mathcal{D}$ (the original universe), which we further train (fine-tune) with σ samples for an increasing number of epochs e (from 100 to 700). The procedure is very similar to *transfer learning* [150]. Figure 5.13d shows the results of this fine tuning for different combinations of σ and e. If we look at the rows of Figure 5.13d, we see that the respective images are more similar to Fig. 5.13a than the image (c) corresponding to the same row. As we increase σ and e, these images become increasingly more similar to the ground truth. For instance, we see that the fine-tuned network can already capture the green spike detail marked in blue in Fig. 5.13a from $\sigma = 500$, $e = 500$, as shown by the red circles in image (d). When training from scratch, this detail requires $\sigma = 1000$ samples to become visible for UMAP, and cannot be captured even for this sample count for the t-SNE projection. Hence, we conclude that we can mimic the ground-truth projection of $\mathcal{D}'$ better by fine-tuning a network pre-trained on a different universe $\mathcal{D}$ than training from scratch on $\mathcal{D}'$ with the same number of samples.

We can draw the following conclusions from this experiment:

- Directly extrapolating training from a universe $\mathcal{D}$ to a different universe $\mathcal{D}'$ will not give good projection results;
- Fine-tuning an existing training on $\mathcal{D}$ with a small number of samples drawn from $\mathcal{D}'$ is possible and can lead to results close to the ground-truth projection of data from $\mathcal{D}'$.

We should stress that the above experiment only hints the possibility of transfer-learning-like training of projections. We do not have enough evidence to assess how much additional training data (from $\mathcal{D}'$) and training time is needed, in general, when extrapolating between two different universes. Moreover, the relationship between the additional training data and training effort required to reach a certain similarity to the ground-truth projection and the similarity of the universes $\mathcal{D}$ and $\mathcal{D}'$ is yet unknown. We consider this to be an interesting topic for future work.

5.5 DISCUSSION

We next discuss how our proposal meets the requirements introduced in Section 5.1.

Quality (C1): We showed that our method achieves very similar quality (measured by five established metrics in dimensionality reduction) to projections well-known to perform well in this area on six challenging multidimensional datasets having up to thousands of dimensions, which are often used as benchmarks in machine learning. Visual comparison also shows that our projections are very close to those computed by existing methods, which, as discussed already, are slower and harder to configure.

Scalability (C2): Even when considering training, our method is roughly one order of magnitude faster than t-SNE and roughly 5 times faster than UMAP for more than

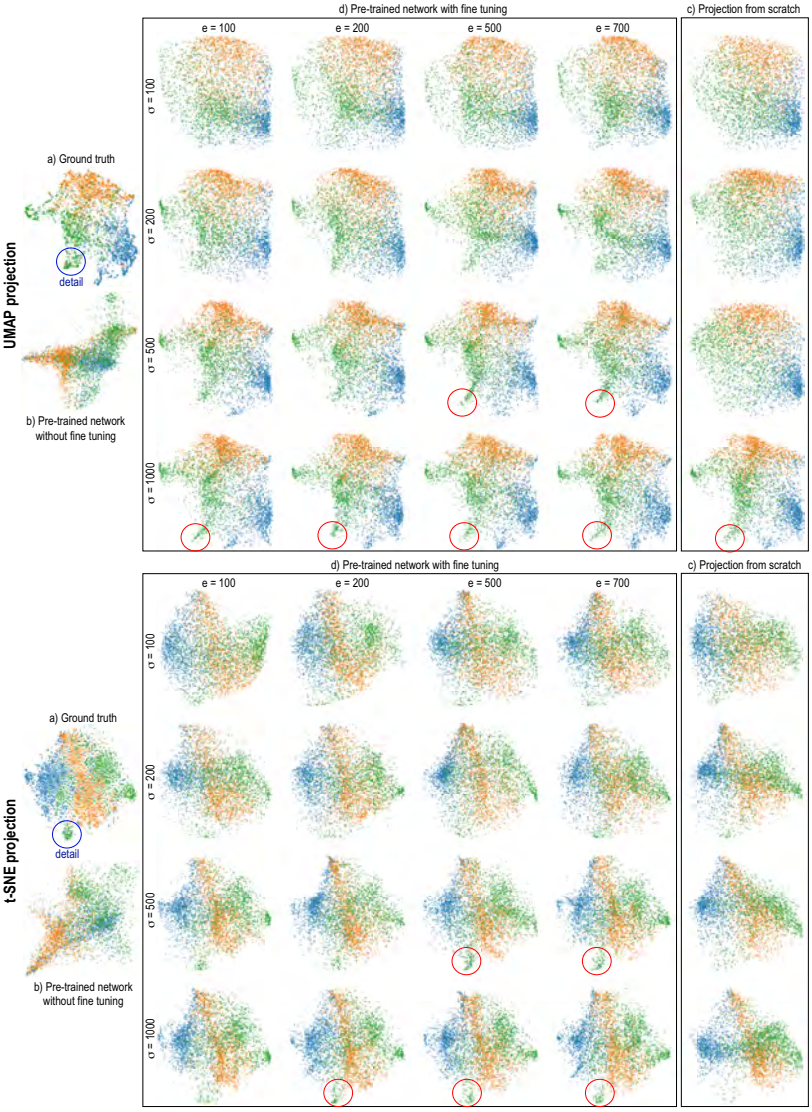


Figure 5.13: Projecting data (4K samples) by fine tuning pre-trained networks mimicking UMAP and t-SNE. a) Test projection. b) Inference by pre-trained network without any fine tuning. Training uses 2K samples from universe $\mathcal{D}$. c) Projections made by our method trained from scratch from σ samples from the new universe $\mathcal{D}'$. d) Projections made by with fine-tuning the pre-trained network with varying numbers of training epochs e and using different numbers σ of samples from $\mathcal{D}'$. Red markers show how the fine-tuned network can already capture a ground-truth detail (shown separately in blue) with fewer training samples σ than when training from scratch with the same σ.

roughly 30K samples. As explained in Section 5.4.4, this is a worst case, since one typically trains once and infers many times on a given data universe. For such cases, our method is more than three to four *orders of magnitude* faster than t-SNE and UMAP, and allows projecting data of millions of samples in a few seconds. The complexity of our method is linear in the number of observations and dimensions. Besides t-SNE and UMAP, our method is actually also faster than other projection methods such as Isomap, LAMP, and MDS.

Ease of use (C3): During inference, our method simply executes a trained neural network, which requires no parameter setting. There is no need for guessing the 'right' values of parameters such as t-SNE's perplexity [214]. During training, the only free parameter to be set is the maximal loss or, alternatively, number of training epochs. The two are related, see Section 5.4.1. As also explained there, a preset of 200 training epochs yielded a loss of 0.005, *i.e.*, practical convergence, for all examples we considered.

Genericity (C4): Our method can learn the behavior of any type of projection technique. We provided examples in Section 5.4 showing this for t-SNE, LAMP, UMAP, MDS, Isomap, LLE, PCA, and pt-SNE. All that is needed to learn is a number of samples from the data universe of interest, represented as n-dimensional feature vectors, and their 2D coordinates computed by the desired projection technique. No other aspects or parameters of the training or inference process are projection-technique specific – that is, projections to be learned can be seen as black boxes. Moreover, no restrictions exist in terms of the dimensionality n of the input feature space in which the data is represented and/or the dimensionality q of the projected data. While we demonstrated our approach only for $q = 2$ (2D projections), which are the most commonly used in infovis, producing higher-dimensional, *e.g.*, 3D projections [38], is equally easy. Such projections are preferred in certain cases as they can preserve the original data structure better than 2D projections [38, 178]. Of course, for our method to be usable, data should come as n-dimensional feature vectors. This is direct *e.g.* in the case of tabular data or images, as discussed in the examples in the paper; for other data types, such as text or videos, suitable feature extraction methods should be used. This is however not a limitation of our method as opposed to other dimensionality reduction methods. Separately, we note that we have only considered projecting quantitative data so far. However, extending our approach to handle categorical data is straightforward by using *e.g.* one-hot encoding or similar techniques [162].

Stability and out-of-sample support (C5): These two issues are strongly interconnected, and actually also connected with the question of how far our networks can generalize what they learn. Let us detail. As outlined in Section 5.3, we take a training set D_s which is supposed to represent well the overall data distribution in a given so-called data universe $\mathcal{D}$, *i.e.*, datasets related to a particular application, such as all handwritten digits, all human face images, all patients in a given population, all street views, and similar. Our approach learns how to project data in $\mathcal{D}$ based on training projections of data in D_s. Hence, the better D_s represents the variability of data in $\mathcal{D}$, the better will our projections mimic actual projections of the same data. Given that neural network inference works deterministically, out-of-sample support is stable in the sense that the same data items (in a dataset $D \subset \mathcal{D}$) are projected to the same locations, which is not the case for many projection methods such as t-SNE, UMAP, and LAMP, to mention just a few. Separately, given the dense structure of the fully-connected network we use (which averages activations from multiple units in an earlier layer to determine

those of the current layer), our approach is stable in the sense that small changes in an input dataset yield only small changes in the resulting projection (see example in Section 5.4.3). Again, this result is far from evident for many existing projection techniques.

Limitations: Our results show that there is a trade-off between the inherent stability and out-of-sample support of our method (discussed above) and the quality (in terms of cluster separation) of the resulting projections. Compared to t-SNE and UMAP, our projections show fuzzier, or less sharply separated clusters. Compared to the other tested projections (MDS, Isomap, LLE, LAMP, autoencoders, PCA), however, our results are almost identical both visually and in terms of the evaluated quality metrics. This trade-off is needed to provide stability: Our method cannot project samples as 'freely' as *e.g.* t-SNE, since it needs to behave deterministically, like any parametric DR technique; on the other hand, this ensures that the same location in nD space projects to the same place in qD space, which is not the case for t-SNE or any other non-parametric DR technique. A similar trade-off between stability and quality exists actually also for dynamic (t-SNE) projections [166] and also for parametric t-SNE [124]. Note that, if desired, we can reduce fuzziness (or, more formally, achieve a higher fit of the learned representation with the training data) by increasing the number of training epochs, decreasing the training loss, or similar well-known techniques in machine learning. However, this is undesired as it can quickly lead to *overfitting*, *i.e.*, it will create suboptimal projections from data which is very different from the training set.

Relation to autoencoders: Both our method and autoencoders use deep learning to perform dimensionality reduction and are parametric techniques. However, the similarities end here: Our method learns from a 2D projection (scatterplot) provided by a user-chosen projection technique; in contrast, autoencoders train with the n-dimensional data itself. Autoencoders propose an *own* embedding of the high-dimensional data into 2D. In contrast, we *learn* whichever embedding was provided to us by the training projection.

Generalization: Related to the last point above, the question arises of *how far* can our approach generalize, or, how densely do we need to sample an universe $\mathcal{D}$ by the training set D_S to create good projections. This is an open question in machine (and deep) learning in general. Yet, we can make the following practical points. First, for the types of (non-trivial) data universes we consider in our evaluations, a few thousands of samples yield already high-accuracy results. Secondly, the larger $\mathcal{D}$ is, the larger (and better spread) the training set D_S needs to be. Section 5.4.5 outlines the limits of this extrapolation: The farthest away is $\mathcal{D}$ spread from the training set D_S, the more will our projections differ from the actual ground-truth projections obtained using classical projection methods. Again, this is not a surprise, but a well-known fact in machine learning. We argue that this is not a problem *in practice* when using projections. Indeed: In all cases we are aware of, researchers typically work for a reasonable amount of time on a *given*, and fixed, data universe $\mathcal{D}$. Hence, they can once train a network from a comprehensive $D_S \subset \mathcal{D}$, after which they can use the network with no changes for data in the same $\mathcal{D}$. Moreover, for cases where one targets a new data universe $\mathcal{D}'$, for which obtaining a comprehensive training set D_S is expensive, the transfer-learning-like approach in Section 5.4.5 can be used. As shown in that Section, one can fine-tune a pre-trained network (on widely available data from a related universe $\mathcal{D}$) with as few as

hundreds of samples from $\mathcal{D}'$.

5.6 CONCLUSION

We have presented NNP, a new method for creating projections of high-dimensional data using a machine learning approach. Based on a small number of 2D projections of a subset of samples from a given data universe, obtained using any user-chosen projection technique, we train a neural network to mimic the 2D projection output, and next use the network to infer projections of unseen data from the same universe. Our method can mimic the quality and visual style of a wide range of established projection techniques, including the well-known visual cluster separation provided by SNE-class methods; is orders of magnitude faster than such methods; has a single parameter for training (with documented preset), and no parameters for inference; can handle datasets of any (quantitative) kind and dimensionality; and delivers inherent stability and out-of-sample support. Our method is simple to implement, requiring only generic (and easily available) software for neural networks, and we provide its source code for replication and actual usage. We show how our approach yields good trade-offs between quality (on the one side) and speed, ease of use, genericity, generalizability, and stability (on the other side).

Many future work directions are next possible. First, we consider generalizing our approach to compute stable projections of dynamic (time-dependent) high-dimensional data and also mixed quantitative-and-qualitative data. Secondly, we consider using different network architectures, cost functions, and training procedures for more accurate handling of more complex data universes. Last but not least, we consider more refined approaches to tackle the transfer learning problem for generalizing learning from a given number of jointly considered data universes and projection techniques.

In Chapter 6, we present a detailed study of how architectural and hyperparameter choices influence the behavior of NNP, and we introduce K-Nearest-Neighbors Projection, which improves on NNP by using local neighborhood information during the training phase.

6 IMPROVING DEEP LEARNED PROJECTIONS

6.1 INTRODUCTION

In Chapter 5 we presented NNP [52], which tries to address the issues of scalability, stability and out-of-sample capability found in popular techniques such as t-SNE. NNP uses a deep learning approach to achieve that: a fully-connected *regression* neural network trained from a small subset of samples of a high-dimensional dataset and their corresponding 2D projection (produced by any DR technique). Next, the network can infer the 2D projection on any high-dimensional dataset drawn from a similar distribution as the training set. This method is much faster than the underlying DR technique (orders of magnitude faster than t-SNE), works deterministically, thereby providing out-of-sample capability by construction, and is simple to implement.

However, it is visible in the examples from Chapter 5 and in Figure 6.1, that the NNP projections have some amount of *diffusion*, *i.e.*, they separate similar-sample clusters less clearly than in a ground-truth t-SNE projection. While visible, it is unclear (i) how much diffusion affects the quality of a projection; (ii) how it depends on hyperparameter settings; and (iii) how one can reduce it.

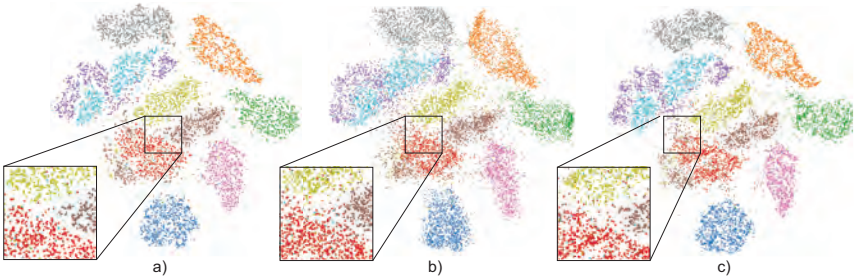


Figure 6.1: Example of diffusion introduced by NNP. (a) Ground-truth t-SNE projection (b) Inferred NNP (10K samples) with diffusion. (c) Inferred KNNP (10K samples) showing less diffusion.

Diffusion can have several causes, *e.g.*: (1) underfitting, by training for too few epochs or having too little data to learn from; (2) overfitting, by not having proper regularization or also by having too little data to learn from; (3) imperfect optimization, with the optimizer getting stuck in local minima. Recently, Espadoto *et al.*[51] studied the causes of diffusion by exploring the *hyperparameter space* of NN projections, showing how these influence the results' quality, gauged by the projection quality metrics. They showed that NNP is *stable* with respect to hyperparameter settings, thereby completing the claim made in Chapter 5 that they can be reliably used for out-of-sample and noisy-data contexts. However, they did not propose a way to reduce diffusion.

This chapter is based on the book chapter (to be published) *Improving Deep Learning Projections by Neighborhood Analysis*. T. S. Modrakowski, M. Espadoto, A. Falcão, N. S. T. Hirata, A. C. Telea.

In this chapter, we extend NNP by proposing a novel approach to deep learning projections. Rather than learning from a *single* sample at a time, we project whole groups of related (neighboring) samples at a time. This aids the network to learn how to preserve neighborhoods. To do this, we explore different schemes of efficient nearest-neighbor search in high-dimensional data during both training and inference. We evaluate our method, called K-Nearest-Neighbors Projection (KNNP) against NNP on a variety of datasets and using several quality metrics, and show that our strategy is both computationally scalable and also leads to quality improvements.

The structure of this chapter is as follows. Section 6.2 discusses related work on DR and neural networks. Section 6.3 details our experimental setup for the NNP evaluation. Section 6.4 presents our results for optimizing NNP, which are next discussed in Section 6.5. Section 6.6 presents the new KNNP Projection method, whose results are discussed next in Section 6.7. Finally, Section 6.8 concludes the chapter.

6.2 RELATED WORK

Related work can be split into dimensionality reduction, deep learning in general, and deep learning projections, as follows.

Dimensionality reduction: Many Dimensionality Reduction (DR) techniques have been developed over the years, with different trade-offs of ease of use, scalability, distance preservation, and out-of-sample capability. Please refer to Chapter 3 for a thorough discussion of dimensionality reduction techniques.

Deep learning: Building well-performing neural network (NN) architectures is very challenging due to the many degrees of freedom allowed by their design process. We outline below five typical such degrees of freedom.

Network Architecture: Part of the power of NNs comes from having many possible architectures, in particular regarding number of layers and layer sizes, if we restrict ourselves to fully-connected networks. There is not a one-size-fits-all set of guidelines to architecting NNs, which are typically created empirically.

Regularization: NNs can be prone to overfitting, which makes them fail to generalize during inference for unseen data. Regularization techniques try to address this by making the learning process harder, so the NN can train for more epochs and generalize better. Regularization techniques include L_2, L_1, max-norm, early stopping, and data augmentation. The L_k regularization techniques, also known as *weight pruning* ($k = 1$) and *weight decay* ($k = 2$), work by adding a penalization term of the form $\lambda \|\boldsymbol{w}\|^k$ to the NN loss function, which equals the k-norm of the weights $\boldsymbol{w}$ of a selected network layer. The parameter λ controls the amount of regularization.

L_1 [152] regularization decreases layer weights with the less important getting down to zero, leading to models with sparse weights. L_2 [107] regularization decreases layer weights to small but non-null values, leading to models where every weight only slightly contributes to the model. Both regularization techniques were observed to help prevent overfitting. Max-norm [193], originally proposed for Collaborative Filtering, was successfully applied as a regularizer for NNs. It imposes a maximum value γ for the norm of the layer weights. This way, weight values are kept under control, similarly to L_2, but using a hard limit. Early stopping [225] is a simple but effective way to prevent overfitting, especially when combined with the other regularization techniques outlined

above. The idea is to stop training when the training loss J_T and validation losses J_V diverge, *i.e.*, J_T keeps decreasing while J_V stops decreasing or worse, starts increasing, both of which signal overfitting.

Optimizers: Different optimizers are used for minimizing the non-linear NN cost. The most used optimizers are based on Mini-batch Stochastic Gradient Descent (SGD) which is a variant of classic Gradient Descent (GD). GD aims to find optimal weights $\boldsymbol{w}$ (that minimize the error created by the NN for the training set) by adjusting these iteratively via the gradient ∇J of the loss function J with respect to $\boldsymbol{w}$ as

$$\boldsymbol{w}_t = \boldsymbol{w}_{t-1} - \eta \nabla J \tag{6.1}$$

where the learning rate η controls how large is each update step M_t. Classic GD uses all data samples at each iteration of Equation (6.1), which is costly for large datasets. Stochastic Gradient Descent (SGD) alleviates this by using only one sample, picked randomly, at each iteration. This speeds up computation at the cost of having to solve a harder problem, as there is less information available to the optimizer. A commonly used optimizer today is Mini-batch SGD, which uses one batch of samples for each GD iteration, thus achieves a compromise between classic GD and SGD. The Momentum method [163] improves SGD convergence by adding to the update vector $\mathbf{u}_t = \eta \nabla J$ at step M_t a fraction ν of the previous update vector $\mathbf{u}_{t-1}$ (Equation (6.2)), *i.e.*

$$\boldsymbol{w}^t = \boldsymbol{w}_{t-1} - (\nu \mathbf{u}_{t-1} + \mathbf{u}_t) \tag{6.2}$$

However, tuning the learning rate η is not trivial: Too small values may make the NN take too long to converge; conversely, too high values may make training miss good minima. Adaptive learning optimizers, such as Adaptive Moment Estimation (ADAM) [102], alleviate this by using squared gradients to compute the learning rate dynamically. This greatly improves convergence speed. However, [222] found that ADAM can find solutions that are worse than those found by Mini-batch SGD. Solving this problem is still an open research question.

Data augmentation: Such techniques generate data that are similar, but not identical, to existing training data, to improve training when training-sets are small. Such techniques highly depend on the data type. Augmentation can be used for regularization since adding more training data creates models that generalize better.

Loss functions: Finally, one needs to select an appropriate loss function J. For regression problems, commonly used loss functions are Mean Squared Error (MSE), Mean Absolute Error (MAE), logcosh, and Huber loss. Table 10 shows the definitions of these functions, where $\hat{\mathbf{y}} = \{\hat{y}_i\}$ is the inferred output vector of the NN and $\mathbf{y} = \{y_i\}$ is the training sample $\hat{\mathbf{y}}$ should match. MSE and logcosh are smoother functions which are easier to optimize by GD or similar methods outlined above. MAE is harder to optimize since its gradient is constant. The Huber loss is somewhere in between the above, according to the parameter α: For values of α near zero, Huber behaves like MAE; for larger α values, it behaves like MSE. MAE and Huber losses are known to be more robust to outliers than MSE.

Deep learning projections: Early on, autoencoders [88, 101] were proposed to generate a compressed, low-dimensional representation on their bottleneck layers by training the network to reproduce its inputs on its outputs. Typically, autoencoders produce results comparable to PCA. The ReNDA algorithm [15] uses two networks, improving on

Table 10: Typical NN loss functions.

Function	Definition
MSE	$\frac{1}{n}\sum_{i=1}^{n}(y_i-\hat{y}_i)^2$
MAE	$\frac{1}{n}\sum_{i=1}^{n}\lvert y_i-\hat{y}_i\rvert$
logcosh	$\frac{1}{n}\sum_{i=1}^{n} log(cosh(y_i-\hat{y}_i))$
Huber	$\begin{cases}\frac{1}{2}(y-\hat{y})^2 & \text{if} \lvert y-\hat{y}\rvert \le \alpha \\ \alpha\lvert y-\hat{y}\rvert-\frac{1}{2}\alpha & \text{otherwise}\end{cases}$

earlier work from the same authors. One network is used to implement a nonlinear generalization of Fisher's Linear Discriminant Analysis, using a method called GerDA; the other network is an Autoencoder used as a regularizer. The method scores well on predictability and has out-of-sample capability. However, it requires labeled data, which may not be always available. Parametric t-SNE (pt-SNE) [124] was proposed to address the out-of-sample limitation of t-SNE. Being of parametric nature (mapping the entire nD input space to the lower-dimensional qD space), it allows out-of-sample behavior by construction. Only few other DR methods are parametric and thus have this ability (*e.g.*, PCA [97], NCA [74], autoencoders [88]).

Lastly, there is NNP, as described in Chapter 5, which is simple to implement, generically learns any projection P for any dataset $D \subset \mathbb{R}^n$, has deterministic (thus, out-of-sample) behavior, and is orders of magnitudes faster than classical projection techniques, in particular t-SNE.

6.3 NNP EVALUATION

While NNP has several advantages, as mentioned in Chapter 5, its quality and stability *vs* parameter setting has not yet been assessed in detail. We address these issues by performing a set of experiments that explore the design space of NNP (Section 6.3.1). For each experiment, we evaluate NNP using several quality metrics (Section 2.2).

6.3.1 *Parameter space exploration*

For each experiment, we train and test the NN projection with various combinations of datasets and hyperparameter settings, and then interpret the obtained results both quantitatively and qualitatively. All parameter values used for each experiment are detailed in this section. Early stopping was used on all experiments, stopping training if the validation loss stops decreasing for more than 10 epochs. Except when noted otherwise in different experiments, the optimizer used was ADAM and the loss function was MSE. As dataset, we use MNIST [112] with 10K test-set samples, and varying training-set sizes of 2K, 5K, 10K and 20K samples. We chose this dataset as it is complex, high-dimensional, has a clear class separation, and it is well known in dimensionality reduction literature. MNIST was also used in the original evaluation of NNP (Section 5.4), which makes it easy to compare our results.

In line with the design choices available to NNs outlined in Section 6.2, we explore the performance of NN projections in the following directions.

Network Architecture (Section 6.4.5): We selected three sizes of neural networks with 360 (small), 720 (medium) and 1440 (large) total number of units, and distributed them

into three different layouts, namely straight (st), wide (wd) and bottleneck (bt), thus yielding nine different architectures, as follows:

- *Small - straight*: 120, 120 and 120 units;
- *Small - wide*: 90, 180 and 90 units;
- *Small - bottleneck*: 150, 60 and 150 units;
- *Medium - straight*: 240, 240 and 240 units;
- *Medium - wide*: 180, 360 and 180 units;
- *Medium - bottleneck*: 300, 120 and 300 units;
- *Large - straight*: 480, 480 and 480 units;
- *Large - wide*: 360, 720 and 360 units;
- *Large - bottleneck*: 600, 240 and 600 units.

Besides these, we also tested the original NNP architecture (See Figure 5.2), called next *Standard*, which has three fully-connected hidden layers, with 256, 512, and 256 units respectively. All architectures use ReLU activation functions, followed by a 2-element layer which uses the sigmoid activation function to encode the 2D projection.

Regularization (Section 6.4.1): We explored the following regularization techniques:

- L_1 with $\lambda \in \{0, 0.001, 0.01, 0.1\}$ with 0 meaning no regularization;
- L_2 with $\lambda \in \{0, 0.001, 0.01, 0.1\}$ with 0 meaning no regularization;
- Max-norm constraint, with $\gamma \in \{0, 1, 2, 3\}$, with 0 meaning no constraint;

Optimizers (Section 6.4.2): We studied two optimizers: ADAM and Mini-batch SGD with learning rates $\eta \in \{0.01, 0.001\}$ and momentum $\nu = 0.9$. In both cases, we set the batch size at 32 samples.

Data augmentation (Section 6.4.3): We explored two data augmentation strategies:

- *Noise Before:* We add Gaussian noise of zero mean and different standard deviations $\sigma \in \{0, 0.001, 0.01\}$, with 0 meaning no noise, to the high-dimensional training data, project this entire (noise + clean samples) dataset, and ask the NN to learn the projection. The idea is that, if the projection to learn (t-SNE in our case) can successfully create well-separated clusters even for noisy data, then our NN should learn how to do this as well;
- *Noise After:* We create the training projection from clean data. We next add Gaussian noise (same σ as before) to the data and train the NN to project the entire (noise + clean samples) dataset to the clean projection. The aim is to force the NN to learn to project slightly different samples to the same 2D point.

Loss functions (Section 6.4.4): We studied four types of loss functions: Huber, with parameters $\alpha \in \{1, 5, 10, 20, 30\}$; Mean Squared Error (MSE), used in the original evaluation of NNP (Section 5.3); Mean Absolute Error (MAE); and logcosh.

Adding more data: For all above directions, we use different training-set sizes (2K, 5K, 10K and 20K samples) to evaluate how this affects the results – that is, how the NN projection quality depends on *both* hyperparameter values and training-set size.

6.4 NNP EVALUATION RESULTS

Table 11: Effect of **regularization**. Rows show metrics for t-SNE (*GT* row) *vs* NN projections using different training-set sizes. Bold shows values closest to *GT*.

a) L_1 **regularization**

Model	λ	M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	0	**0.705**	**0.843**	**0.957**	**0.443**	50	6.20
	0.001	0.677	0.827	0.948	0.439	58	7.14
	0.01	0.660	0.815	0.945	0.438	94	10.93
	0.1	0.632	0.806	0.943	0.454	82	9.98
5K	0	**0.738**	**0.871**	**0.962**	**0.423**	26	7.22
	0.001	0.692	0.845	0.953	0.436	38	10.05
	0.01	0.670	0.835	0.947	0.427	68	18.29
	0.1	0.599	0.815	0.945	0.459	53	14.58
10K	0	**0.834**	**0.902**	**0.968**	**0.337**	45	22.32
	0.001	0.753	0.852	0.958	0.348	31	16.09
	0.01	0.722	0.833	0.951	0.352	39	19.12
	0.1	0.665	0.811	0.947	0.345	61	30.67
20K	0	**0.885**	**0.922**	**0.967**	**0.341**	49	47.28
	0.001	0.816	0.883	0.960	0.364	30	29.35
	0.01	0.743	0.842	0.954	0.366	28	26.89
	0.1	0.707	0.822	0.946	0.364	25	24.17

b) L_2 **regularization**

M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
0.929	0.990	0.976	0.277		
0.695	0.839	0.956	0.437	35	4.61
0.711	**0.847**	**0.958**	0.432	29	4.27
0.684	0.834	0.954	0.433	42	5.57
0.683	0.830	0.952	**0.428**	68	8.54
0.767	**0.880**	**0.963**	0.422	53	14.33
0.742	0.875	**0.963**	0.419	28	7.71
0.733	0.866	0.959	**0.416**	55	15.24
0.709	0.860	0.958	0.429	45	12.51
0.833	**0.899**	**0.967**	0.342	43	20.93
0.821	**0.899**	0.966	0.340	55	27.88
0.798	0.880	0.963	0.337	34	17.37
0.773	0.865	0.961	**0.336**	36	18.48
0.885	**0.922**	**0.967**	0.341	46	43.49
0.870	0.915	0.966	0.343	34	33.06
0.853	0.902	0.963	0.344	40	38.05
0.826	0.883	0.960	**0.339**	38	37.87

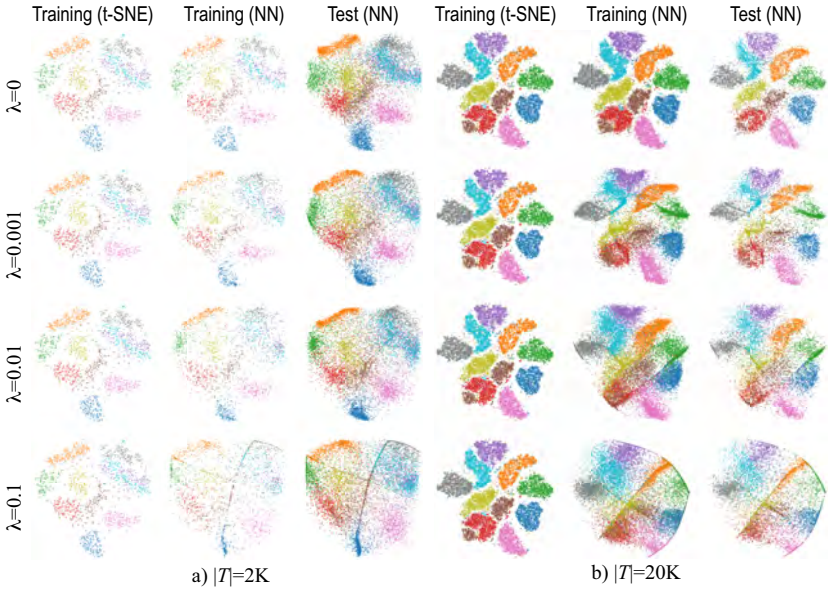


Figure 6.2: L_1 **regularization:** Effect of λ for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set.

We next present the details of each experiment along with the obtained results.

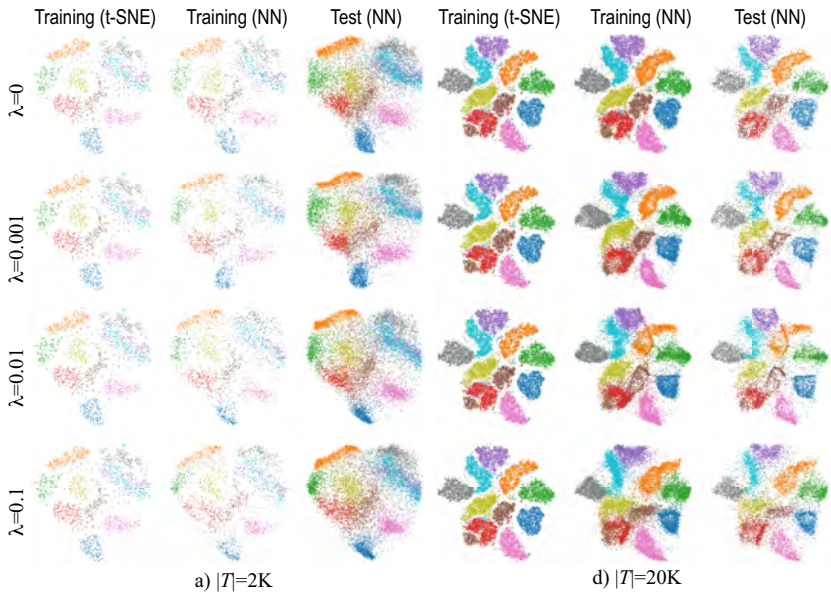


Figure 6.3: L_2 **regularization:** Effect of λ for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set.

6.4.1 *Regularization*

We use increasing amounts of L_1 and L_2 regularization to test if, by having a penalty term on the cost function during training, the NN can generalize better. We use L_1 and L_2 separately, to study how their effects compare to each other.

Figures 6.2 and 6.3 show the results. We see that, for both L_1 and L_2, the higher the regularization values λ, the worse are the results. For instance, the resulting projection (train or test) becomes completely unrelated to the ground truth (training t-SNE projection) when $\lambda = 0.1$. Separately, we see that L_2 regularization performs better – that is, it produces projections which are closer to the ground truth than the corresponding projections produced using L_1 regularization for the same λ values. Table 11 confirms the above visual findings by showing quality metrics for the L_1 and L_2 regularization experiments. Overall, we see that L_2 regularization produces M_{nh}, M_t, and M_c metrics closer to the ground-truth (GT) values than L_1 regularization. This table also shows another interesting insight: The NN projections yield *higher* Shepard correlation M_s values than the ground truth t-SNE projection, for all regularization settings (slightly higher for L_1 than L_2). This tells us that the NN aims to preserve the nD distances in the 2D projection *more* than the t-SNE projection does (see definition of M_s, Table 1). This explains, in turn, the diffusion we see in the NN projections. In contrast, t-SNE does not aim to optimize for distance preservation, but neighborhood preservation, which results in a better cluster separation but lower M_s values.

The rightmost two columns in Tables 11(a,b) show the training effort needed for convergence (epochs and seconds). We see that convergence is achieved *for all cases* in under 70 epochs, regardless of the regularization type (L_1 or L_2) or strength λ. Also, L_1 and L_2 regularization have comparable costs, with L_2 being slightly faster than L_1 for smaller

training datasets. Overall, the above tells us that the NN converges robustly regardless of regularization settings.

Next, we study max-norm regularization, to see how this affects the NN generalization capability. Figure 6.4 shows that the projection quality is not strongly dependent on γ, and the metrics in Table 12(a) confirm this. More importantly, we see that max-norm yields projections which are better in terms of all quality metrics than L_1 and L_2 regularization, and closer to the t-SNE ground truth. Effort-wise, max-norm regularization is very similar to L_1 and L_2 (compare rightmost columns in Table 12 (a) with those in Table 11(a,b)). In conclusion, for this particular problem we determine that regularization brings no clear benefit.

Table 12: Effect of **max-norm** (left) and **optimizers** (right). Metrics shown for t-SNE (*GT* row) *vs* NN projections using different training-set sizes. Bold shows values closest to *GT*.

a) **Max-norm regularization**

Model	γ	M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	0	**0.701**	0.839	0.956	0.443	44	5.45
	1	0.692	0.836	0.956	**0.431**	32	4.47
	2	0.699	**0.842**	**0.957**	0.441	45	5.80
	3	0.698	0.837	0.956	0.441	31	4.47
5K	0	**0.759**	**0.881**	**0.964**	**0.417**	51	13.34
	1	0.756	0.880	**0.964**	0.421	40	10.70
	2	0.740	0.866	0.961	0.420	24	7.05
	3	0.755	0.879	0.963	0.423	48	13.23
10K	0	0.824	0.898	0.966	**0.337**	37	18.14
	1	**0.840**	0.904	0.967	0.338	31	15.43
	2	0.829	0.903	0.967	0.340	37	18.67
	3	0.837	**0.905**	**0.968**	0.338	53	26.63
20K	0	**0.886**	**0.923**	**0.967**	0.342	56	52.65
	1	0.870	0.918	0.967	**0.340**	26	25.22
	2	0.881	0.917	0.967	0.341	30	28.88
	3	0.879	0.920	0.967	0.345	34	34.11

b) **Optimizers**

Model	Optimizer (η)	M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	ADAM	**0.696**	**0.841**	**0.956**	**0.447**	30	3.72
	SGD (0.01)	0.625	0.791	0.938	0.464	97	8.32
	SGD (0.001)	0.610	0.787	0.938	0.464	455	36.56
5K	ADAM	**0.733**	**0.861**	**0.960**	**0.421**	19	5.27
	SGD (0.01)	0.655	0.817	0.945	0.439	86	16.90
	SGD (0.001)	0.641	0.808	0.942	0.443	402	77.17
10K	ADAM	**0.842**	**0.905**	**0.968**	**0.343**	56	26.51
	SGD (0.01)	0.707	0.821	0.949	0.362	75	28.55
	SGD (0.001)	0.690	0.812	0.948	0.360	392	147.60
20K	ADAM	**0.882**	**0.920**	**0.968**	**0.339**	43	40.77
	SGD (0.01)	0.769	0.838	0.952	0.356	129	94.30
	SGD (0.001)	0.754	0.836	0.952	0.370	423	309.19

6.4.2 *Optimizer*

The quality of the NN projection obviously depends on how well the optimization method used during training can minimize the cost function (Section 6.2). To find out how the projection quality is influenced by optimization choices, we trained the NN using the ADAM optimizer with its default settings, and also with SGD with learning rates $\eta \in \{0.01, 0.001\}$. Figure 6.5 shows that the ADAM optimizer produces results with considerably less diffusion than SGD. Table 12(b) clearly confirms this, as ADAM scores better than SGD for all considered quality metrics. We also see here that ADAM converges much faster than SGD. Since, additionally, ADAM works well with its default parameters, we conclude that this is the optimizer of choice for our problem.

6.4.3 *Noise-based data augmentation*

We turn to data augmentation to try to reduce diffusion in the NN projection. For this, we add noise to the data as described in Section 6.3.1 to observe if it improves learning. Figure 6.6 shows that both the *Noise before* and *Noise after* strategies produce quite similar results, which are also close to the ground truth. Table 13(a,b) confirms this, additionally

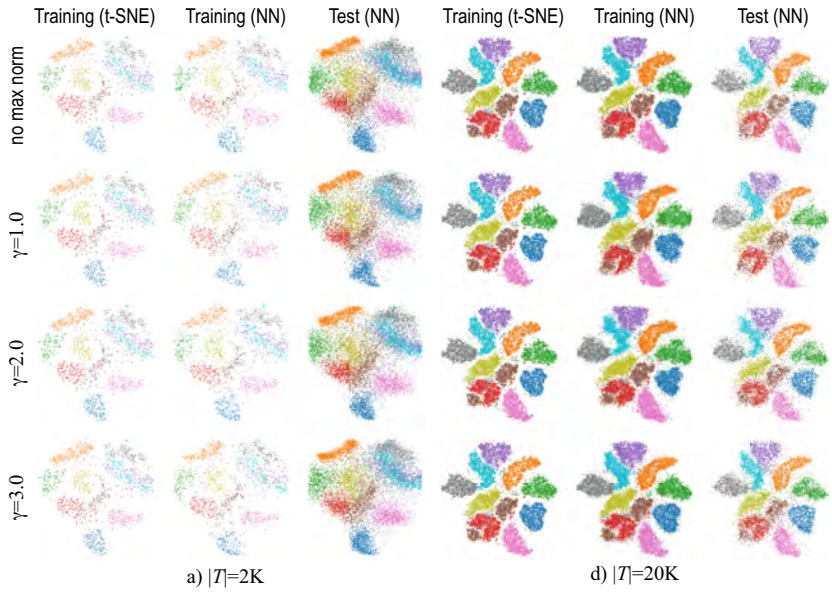


Figure 6.4: **Max-norm:** Effect of γ for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set.

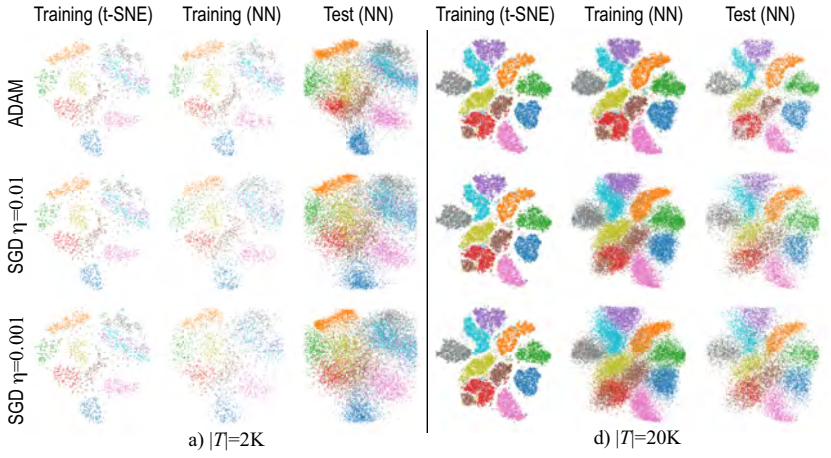


Figure 6.5: **Optimizer:** Effects of different settings (ADAM, SGD with $\eta \in \{0.001, 0.01\}$) for different training-set sizes. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set.

showing that the *Noise after* strategy yields slightly higher quality metrics on average than *Noise before*. More importantly, if we compare these values with those obtained by trying different regularization techniques and optimizers (Tables 11-12), we see that *Noise after* slightly improves the projection quality.

Table 13: Effect of **data augmentation**. Rows show metrics for t-SNE (*GT* row)) *vs* NNP (other rows). Right two columns in each table show training effort (epochs and time). Bold shows values closest to *GT*.

a) **Noise after** strategy

Model	Noise σ	M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	0	0.717	0.846	0.958	0.448	31	8.84
	0.001	0.726	0.852	**0.960**	**0.430**	37	10.57
	0.01	**0.729**	**0.856**	**0.960**	0.433	54	14.52
5K	0	**0.783**	0.892	**0.966**	**0.401**	43	23.28
	0.001	0.780	**0.895**	**0.966**	0.408	52	29.44
	0.01	**0.783**	0.892	**0.966**	**0.401**	47	26.84
10K	0	**0.849**	0.909	**0.968**	0.339	44	44.06
	0.001	0.844	0.909	**0.968**	0.337	36	37.76
	0.01	0.848	**0.910**	**0.968**	**0.333**	59	60.31
20K	0	0.887	0.924	0.966	0.340	55	105.87
	0.001	**0.888**	0.924	**0.967**	**0.336**	46	88.09
	0.01	0.885	**0.925**	**0.967**	0.339	51	97.06

b) **Noise before** strategy

M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
0.929	0.990	0.976	0.277		
0.712	**0.842**	0.957	0.446	23	7.17
0.679	**0.842**	**0.959**	0.422	33	9.57
0.682	0.833	0.956	**0.421**	20	6.86
0.785	**0.894**	0.966	0.401	62	33.34
0.793	0.884	0.966	**0.364**	31	17.91
0.802	0.888	**0.967**	0.366	49	28.68
0.849	**0.908**	**0.968**	0.336	36	35.61
0.798	0.901	0.966	0.304	37	39.58
0.802	0.904	0.966	**0.302**	53	55.58
0.888	**0.925**	**0.967**	**0.337**	40	76.08
0.865	0.920	**0.967**	0.385	42	81.55
0.869	0.920	**0.967**	0.392	41	80.52

6.4.4 *Loss function*

Next we study the effect of using different loss functions. Figure 6.7 shows that MAE produces visual clusters that are slightly sharper than the ones created with the other loss functions studied. Also, we see that this effect is more pronounced on tests with lower numbers of training samples. This effect is confirmed by looking at the quality metrics in Table 14 (a): For instance, using MAE yields an increase of M_{nh} from roughly 0.70 (when using the other loss functions) to roughly 0.74 for the smallest test-set of 2K samples; for the largest test-set of 20K samples, the comparable M_{nh} increase is from roughly 0.87 to 0.88. Still, MAE achieves consistently the best quality metrics for almost all the tested cases, as compared to using the other loss functions. Separately, we see in Table 14 (a) that the training effort for MAE is higher than when using the other loss functions. However, as the number of samples increases, the training-effort difference decreases, which is important, as it tells us that, for realistic (larger training-sets) cases, using MAE is not really costing more than using other loss functions. Given the quality increase, we conclude that MAE is the best loss function.

6.4.5 *Network Architecture*

Finally, we study the effect of using different NN architectures. Figure 6.8 shows that the architecture *Large - bottleneck* produces visual clusters that are slightly sharper than the ones created by the other architectures studied. This is confirmed by the quality metrics in Table 14 (b): We see that *Large - bottleneck* has a M_{nh} about 0.04 higher for all training-set sizes. Also, while this architecture is larger than the others, its training effort is quite similar to that of the other architectures.

6.5 INSIGHTS FROM EVALUATION

We next summarize the obtained insights from the evaluation of NNP, as follows.

Optimal settings: Our experiments showed that NNP attains optimal quality (and closest to the ground-truth t-SNE projection) with no regularization, ADAM optimizer, *Noise*

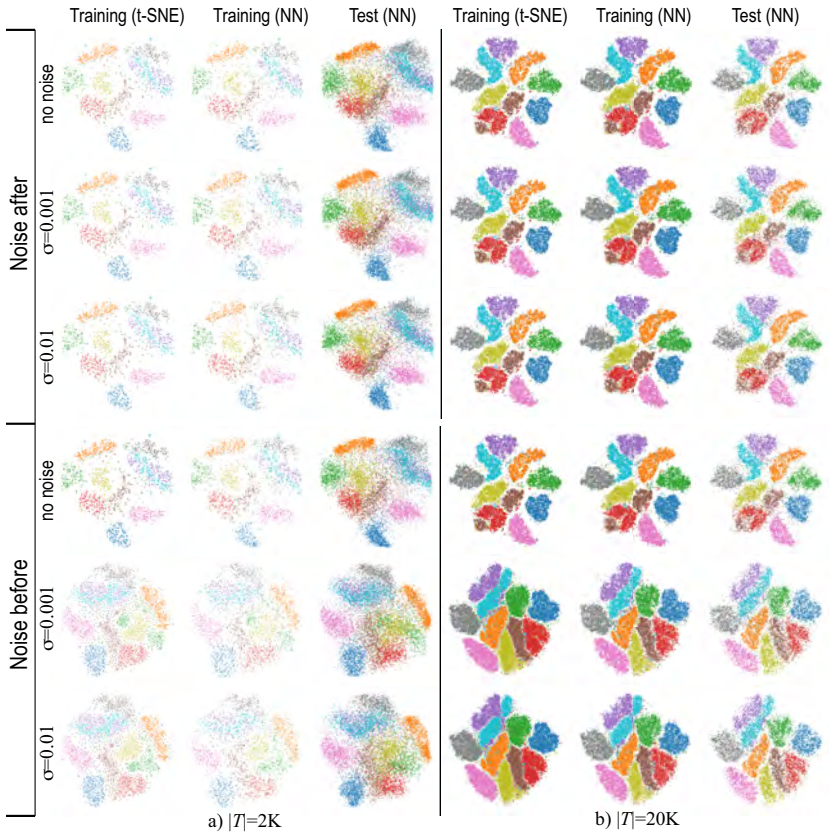


Figure 6.6: **Noise after** and **noise before** data augmentation: Effect of noise strength $\sigma \in \{0, 0.001, 0.01\}$. Compare the ground truth (training-set, projected by t-SNE) with the NN results on the training-set, respectively test-set.

after data augmentation, MAE loss function, and *Large - Bottleneck* architecture. The choice of σ (noise standard deviation for data augmentation) affects very little the measured quality, so one should not be concerned in practice by how to set this parameter.

Given the above optimal settings, with $\sigma = 0.01$, we did one last experiment to evaluate how they perform when *combined*. We ran both the original *Std* architecture and the *Large - Bottleneck*, both using the optimal settings, to better assess the effect of the architecture change. In Table 15 we see that *Large - Bottleneck* performs better than *Std* on practically all metrics and for all training-set sizes. This improvement can be seen even when compared to the best results of each individual test, especially for smaller training-set sizes. Figure 6.9 shows this improvement in the form of less fuzziness and better separated clusters.

Quality: We have measured four well-known projection quality metrics: neighborhood hit, trustworthiness, continuity, and Shepard diagram correlation. Our experiments show that all these metrics are stable with respect to hyperparameter settings. More importantly, the optimal setting outlined above yields values *closer* to the ground-truth (t-SNE) values than the results presented for NNP in Section 5.4. As the training-set

Table 14: Effect of different **loss functions** (left) and **architectures** (right). Rows show metrics for t-SNE (*GT* row) *vs* NN projections using different training-set sizes. Bold shows values closest to *GT*.

a) **Loss Functions**

Model	Loss (α)	M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277		
2K	Huber (1.0)	0.706	0.839	0.956	0.445	34	5.94
	Huber (5.0)	0.687	0.827	0.953	0.447	16	3.99
	Huber (10.0)	0.704	0.839	0.957	0.431	45	7.53
	Huber (20.0)	0.692	0.835	0.956	0.442	32	5.88
	Huber (30.0)	0.695	0.836	0.956	0.433	30	5.98
	logcosh	0.704	0.839	0.957	0.434	33	6.37
	MAE	**0.742**	**0.866**	**0.962**	**0.423**	78	11.05
	MSE	0.704	0.842	0.957	0.442	40	6.86
5K	Huber (1.0)	0.762	0.883	0.964	0.420	50	14.50
	Huber (5.0)	0.745	0.871	0.963	0.426	26	8.34
	Huber (10.0)	0.769	0.886	**0.965**	**0.416**	69	19.79
	Huber (20.0)	0.763	0.884	**0.965**	0.420	62	18.18
	Huber (30.0)	0.768	0.883	**0.965**	0.420	55	16.16
	logcosh	0.768	0.883	**0.965**	0.425	54	16.03
	MAE	**0.781**	**0.893**	**0.965**	0.418	57	16.56
	MSE	0.753	0.874	0.963	0.428	30	9.67
10K	Huber (1.0)	0.831	0.898	0.968	0.338	38	19.56
	Huber (5.0)	0.833	0.902	0.968	0.342	40	20.64
	Huber (10.0)	0.837	0.906	**0.969**	0.344	52	26.98
	Huber (20.0)	0.831	0.900	0.968	0.344	38	19.97
	Huber (30.0)	0.831	0.902	0.968	0.348	36	19.55
	logcosh	0.818	0.893	0.967	0.347	25	14.00
	MAE	**0.848**	**0.912**	0.968	**0.333**	58	29.55
	MSE	0.839	0.906	0.968	0.339	65	32.42
20K	Huber (1.0)	0.856	0.907	0.967	0.353	20	19.57
	Huber (5.0)	0.881	0.918	0.967	0.344	44	41.69
	Huber (10.0)	0.882	0.921	**0.968**	0.344	36	34.48
	Huber (20.0)	0.881	0.920	0.967	0.342	45	43.96
	Huber (30.0)	0.877	0.915	0.967	0.341	29	28.71
	logcosh	0.884	0.919	0.967	**0.335**	35	35.46
	MAE	**0.887**	**0.927**	0.966	0.339	47	44.55
	MSE	0.871	0.914	0.967	0.341	23	21.68

b) **Network Architecture**

Model	NN Arch	M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277	0	0
2K	small st	0.680	0.827	0.951	0.437	30	5.11
	small bt	0.670	0.819	0.950	0.453	18	3.85
	small wd	0.672	0.820	0.949	0.463	17	3.82
	medium st	0.683	0.827	0.952	0.441	17	3.83
	medium bt	0.690	0.833	0.955	0.456	25	4.96
	medium wd	0.702	0.838	0.956	0.438	44	7.17
	large st	0.692	0.835	0.956	0.447	19	4.66
	large bt	**0.720**	**0.852**	**0.961**	**0.430**	50	8.95
	large wd	0.713	0.847	0.959	0.434	45	8.30
5K	small st	0.744	0.875	0.962	0.414	66	18.31
	small bt	0.719	0.855	0.958	0.423	17	5.78
	small wd	0.726	0.864	0.959	0.424	40	12.21
	medium st	**0.761**	0.879	0.963	0.418	42	12.67
	medium bt	0.742	0.872	0.962	0.426	33	10.54
	medium wd	0.740	0.873	0.963	0.419	40	12.53
	large st	0.752	0.874	**0.964**	**0.408**	29	10.95
	large bt	**0.761**	**0.880**	**0.964**	0.420	34	12.02
	large wd	0.755	0.878	**0.964**	0.423	38	13.87
10K	small st	0.818	0.893	0.966	0.338	43	21.02
	small bt	0.820	0.893	0.966	0.330	54	27.35
	small wd	0.794	0.879	0.963	0.330	28	15.35
	medium st	0.828	0.900	0.968	0.343	45	22.74
	medium bt	0.820	0.895	0.967	0.343	31	16.68
	medium wd	0.825	0.899	0.967	**0.338**	49	25.74
	large st	0.831	0.902	0.968	**0.338**	32	20.34
	large bt	**0.836**	**0.905**	**0.969**	0.341	36	21.97
	large wd	0.830	0.900	0.968	**0.338**	30	19.31
20K	small st	0.865	0.910	0.965	**0.335**	30	30.81
	small bt	0.838	0.891	0.965	0.353	16	15.74
	small wd	0.865	0.910	0.965	0.345	37	34.66
	medium st	0.882	0.922	**0.967**	0.339	45	41.97
	medium bt	0.882	0.921	**0.967**	0.340	45	41.35
	medium wd	0.874	0.917	**0.967**	0.346	34	32.92
	large st	0.886	0.924	**0.967**	0.340	45	50.74
	large bt	**0.890**	**0.925**	**0.967**	0.342	37	42.48
	large wd	0.878	0.917	**0.967**	0.345	29	33.03

increases in size, the NNP quality metrics *consistently* approach the ground-truth values – see Tables 11-14 for training-sets from 2K to 20K samples. The difference of the two is under 5% on average for training-sets of 20K points. Visual examination of the NNP projections shows that these exhibit a discernible amount of diffusion as compared to the ground-truth t-SNE projections. While diffusion clearly *decreases* with training-set size, it is still present even for the optimal parameter settings and 20K training samples – compare *e.g.* the inference on unseen data in Figure 6.6(b), Test (NN), with Figure 6.6(b), Training (t-SNE).

Stability: An important result of our experiments is that NNP is stable with respect to training set sizes, hyperparameter settings, noise and loss functions. Indeed, Figures 6.2-6.8 show, regardless from the already discussed diffusion effect, practically the same *shape* and relative *positions* of the data clusters in the test projections (NN method run on unseen data) and the ground-truth t-SNE projections, for all tested configurations. The stability of NNP with respect to training data, parameter settings, and noise is in *stark contrast* with the instability of the ground-truth t-SNE projection with respect to *all*

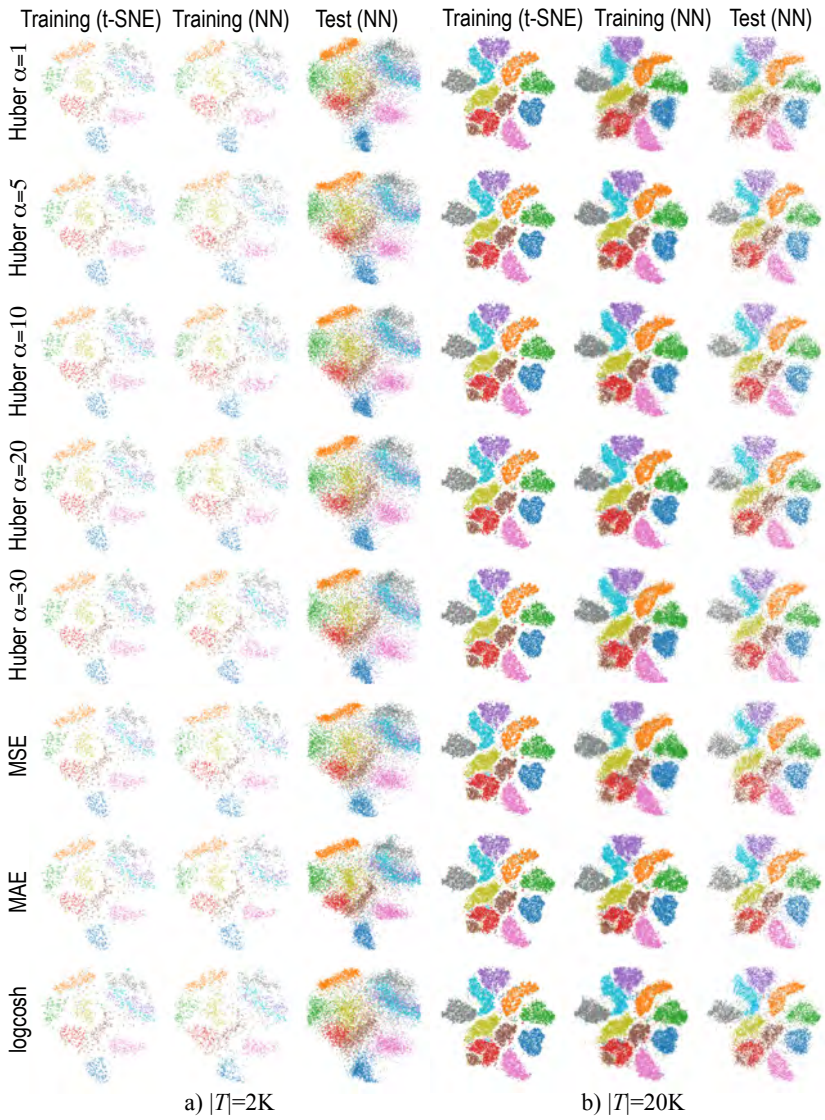


Figure 6.7: **Loss:** Effect of different loss functions. Compare the ground truth (training-set, projected by t-SNE) with the NNP results on the training-set, respectively test-set.

these three factors, and is of important practical added-value in many applications [214].

6.6 IMPROVING NNP BY NEIGHBORHOOD ANALYSIS

Following our analysis of the NNP evaluation (Section 6.5), we see that NNP scores very well on stability and quality consistency with respect to hyperparameter values. In the same time, the quality is still on average 5% lower than that of the ground-truth (t-SNE)

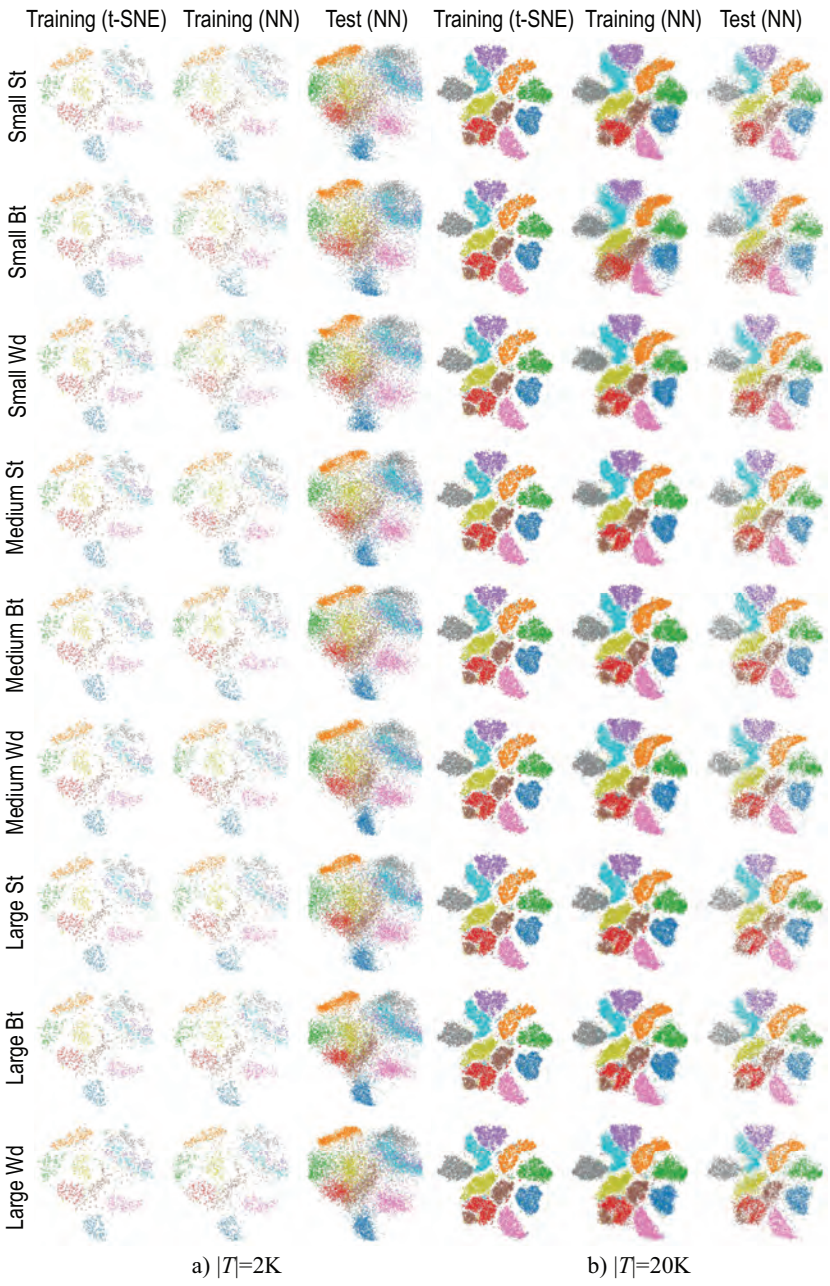


Figure 6.8: **Arch:** Effect of different architectures. Compare the ground truth (training-set, projected by t-SNE) with the NNP results on the training-set, respectively test-set.

projection. This is visible in the still higher diffusion of NNP as compared to t-SNE. Our experiments show that hyperparameter settings, including regularization, data augmen-

Table 15: Effect of using **optimal settings**. Metrics shown for t-SNE (*GT* row) *vs* NN projections using different training-set sizes. Bold shows values closest to *GT*.

Model	NN Arch	M_{nh}	M_t	M_c	M_s	# epochs	Time (s)
GT		0.929	0.990	0.976	0.277	0	0
2K	std	0.753	0.871	0.963	0.433	73	14.58
	large bt	**0.773**	**0.878**	**0.964**	**0.426**	82	18.44
5K	std	0.794	0.904	0.964	**0.411**	129	60.26
	large bt	**0.813**	**0.906**	**0.966**	**0.411**	70	37.19
10K	std	**0.850**	**0.916**	**0.967**	0.334	113	104.39
	large bt	**0.850**	0.913	0.966	**0.331**	108	112.53
20K	std	0.884	0.923	**0.964**	**0.335**	121	215.66
	large bt	**0.891**	**0.929**	**0.964**	**0.335**	101	205.07

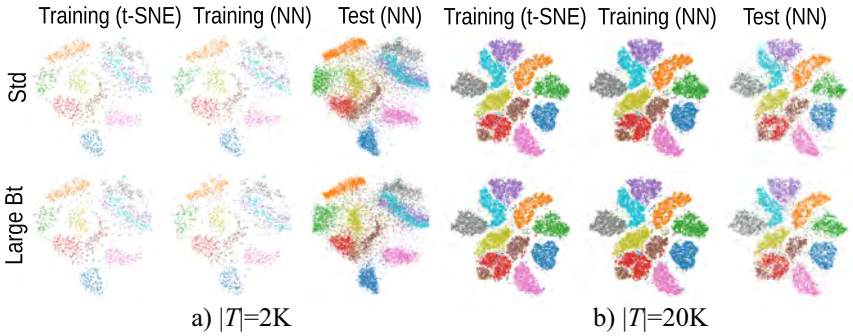


Figure 6.9: Effect of using **optimal settings**. Optimal settings for *Std* and *Large - Bottleneck* NN architectures. Compare the ground truth (training-set, projected by t-SNE) with the NNP results on the training-set, respectively test-set.

tation, optimizer, loss function and network architecture cannot fully eliminate diffusion, although by using MAE as loss function, quality metrics increased in value.

The strong visual separation of data clusters produced by t-SNE is likely one of the most praised feature of this method. t-SNE achieves this by essentially considering the preservation of *neighborhoods* rather than of point-pair distances. We next leverage this intuition in the context of NNP's deep learning approach to projections.

Consider the NNP approach, where each training sample $\mathbf{x}$ is fed into the network with its corresponding ground-truth (t-SNE) coordinate $P(\mathbf{x})$ as a training label. We replace each such training pair $(\mathbf{x}, P(\mathbf{x}))$ with a pair of *neighborhoods* $(\nu(\mathbf{x}), P(\nu(\mathbf{x}))$. Here, $\nu(\mathbf{x})$ are the K nearest neighbors of $\mathbf{x}$ in D; and $P(\nu(x))$ are the ground-truth projections of these neighbors. We compute neighborhoods ν using both a fast approximate nearest-neighbor search [131] and an exact, slower, brute-force search, to check whether the approximate search has any negative impact on quality. We call our new model K-Nearest-Neighbors Projection, or KNNP.

During inference, we compute nearest neighbors over points from the training set. There are two reasons for this: (1) The training set is already learned (known) by the network; (2) The training set is already indexed for fast search [131].

We tune the hyperparameters of the KNNP model following the results in Section 6.5. We use MAE as our loss function, which is averaged over the K neighbors as each one is treated as a single sample or label. We chose ADAM as our optimizer. The architecture of the network follows the one in Section 6.3.1 aside from the input and output layers

which are scaled so that each input layer containing K nD points outputs a single 2D point.

6.7 KNNP EVALUATION

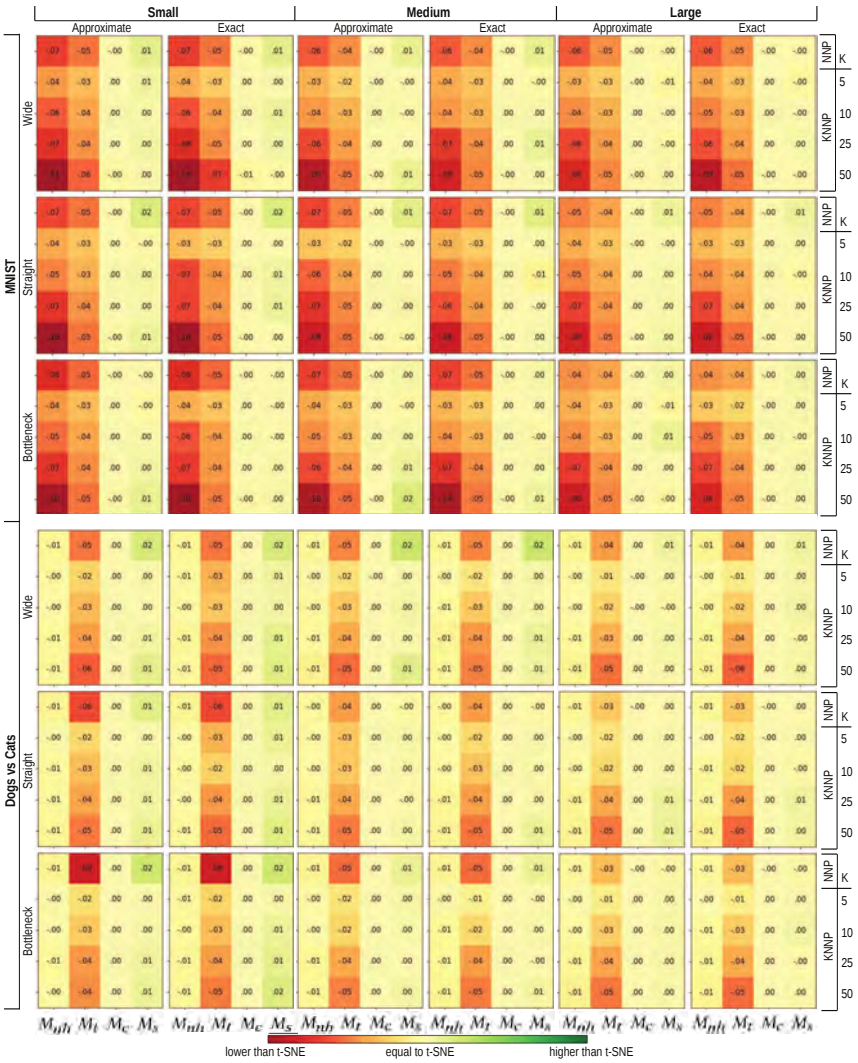


Figure 6.10: Comparison of the difference in four quality metrics M_{nh}, M_t, M_c, and M_s between t-SNE and NNP, respectively t-SNE and KNNP. The comparison is done on the MNIST and Dogs *vs* Cats datasets, for five K values, using both exact and approximate search, for three architecture styles (wide, straight, bottleneck), each having three sizes (small, medium, large). Red colors indicate cases which are farthest below t-SNE's quality.

We next compare the KNNP method introduced in Section 6.6 with the original NNP method using the optimized hyperparameter settings from Section 6.4 and with the

ground-truth t-SNE projection. For this, we use as metrics Turstworthiness (M_t), Continuity (M_c), Neighborhood Hit (M_{nh}) and Shepard diagram correlation (M_s), as defined in Section 2.2. In addition to the MNIST dataset (Section 6.3.1), we use three more datasets to the comparison, namely Fashion MNIST [223], Dogs *vs* Cats [49] and IMDB Movie Review [122]. Please refer to Section 2.3 for a detailed description of the datasets.

We next show the performance of KNNP *vs* NNP and t-SNE, for training data (Section 6.7.1) and test data (Section 6.7.2). We also show how quality depends on the training set size (Section 6.7.3) and evaluate KNNP's speed *vs* other techniques (Section 6.7.4). Finally, we show actual projection plots computed by KNNP, NNP, and t-SNE (Section 6.7.5). Due to space restrictions, we present only a subset of our results.

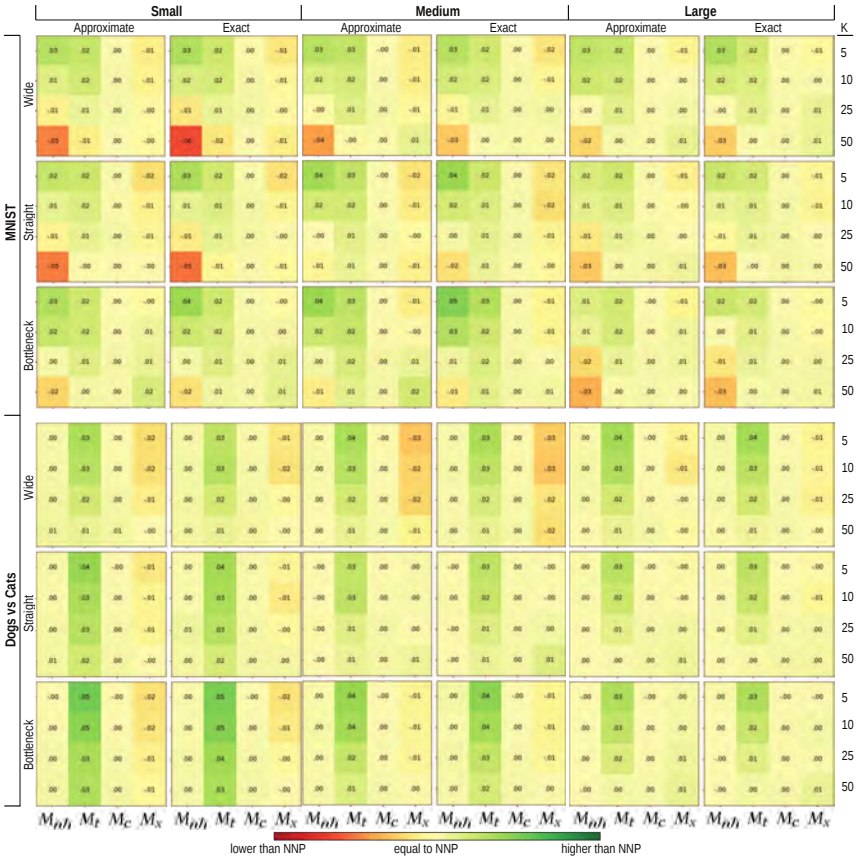


Figure 6.11: Comparison of quality metrics of KNNP *vs* NNP for the same datasets, architectures, and parameters as in Fig. 6.10. Green indicates cases where KNNP performs better than NNP.

6.7.1 *Quality on training data*

Figure 6.10 compares the performance of KNNP, the original method (NNP), and the ground truth (GT, t-SNE) across four quality metrics. Red, yellow, and green indicate that the respective method has a quality lower than, similar to, respectively higher than GT.

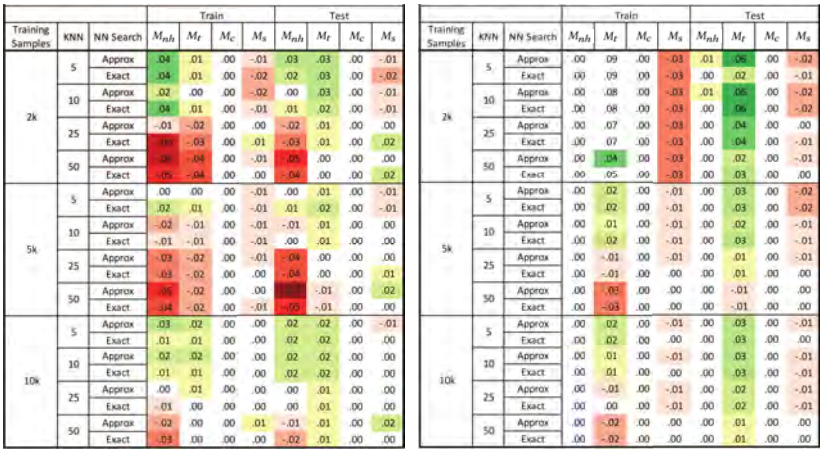


Figure 6.12: Comparison of KNNP *vs* NNP quality metrics for different training set sizes on MNIST (left) and Dogs *vs* Cats (right). Green marks cases where KNNP outperforms NNP.

We see that, for $K = 5$ neighbors, KNNP performs slightly better than NNP, in virtually all cases and for all quality metrics. We also see that quality does not vary much with architecture style or size. Hence, when running on a tight computational budget (where one cannot train or test large architectures), KNNP has a small edge over NNP.

6.7.2 *Quality on testing data*

So far, we compared both deep learning projections (KNNP and NNP) against each other and against the GT (t-SNE). For testing data, we cannot do the latter comparison, since t-SNE is not a deterministic method, and does not have an out-of-sample capability. Hence, for testing data, we next compare KNNP and NNP – trained on the same data, and tested on the same data – against each other only.

Figure 6.11 shows that KNNP gets the largest quality boost *vs* NNP for $K = 5$ neighbors again. As in Fig. 6.10, the style and size of architecture do not influence the results. Overall, KNNP yields better quality than NNP. However, which metric (of the four evaluated) is most improved depends on the dataset. This is expected, since neither NNP nor KNNP do *explicitly* optimize for a given quality metric.

6.7.3 *Quality as function of training set size*

Figure 6.12 shows how the quality of KNNP compares to that of NNP for different training-set sizes. We see that the added-value of KNNP *vs* NNP is higher for fewer training samples, particularly so for $K = 5$ neighbors. Hence, when the user can only use a small training-set, the relative added-value of KNNP *vs* NNP increases.

6.7.4 *Computational scalability*

We next compare the speed of KNNP, NNP, and other well-known techniques for up to 1M test samples. Figure 6.13a shows the projection time (log scale) as a function of the dataset size for *parametric* techniques. We see that all techniques are linear with

dataset size. NNP is the fastest of all compared techniques, with KNNP using approximate nearest-neighbor (ANN) search coming close. Figure 6.13b adds *non-parametric* techniques to the comparison, specifically MDS [206], t-SNE, LSP [154], and LAMP [95].

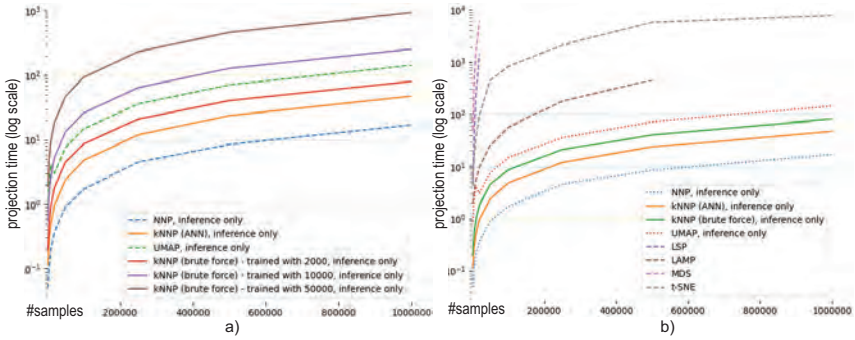


Figure 6.13: Projection times for parametric techniques only (a) and for parametric and non-parametric techniques (b)

We see the same trend as before. Also, we see that KNNP is faster than all non-parametric techniques. Separately, Figure 6.14 shows training time for the parametric techniques for up to 1M training samples. Beyond 250K samples, UMAP failed to finish training. NNP and KNNP with ANN search show basically the same speed, being both faster than KNNP with brute-force search.

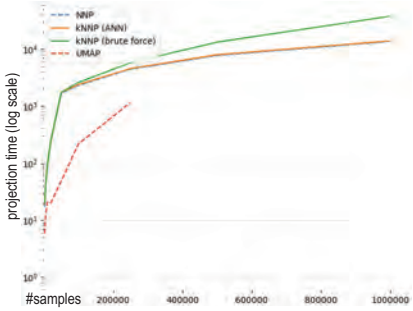


Figure 6.14: Comparison of training times between parametric techniques

6.7.5 *Projection scatterplots*

Figure 6.15 shows samples of scatterplots created with t-SNE, NNP, and KNNP with ANN and brute force search. We see that KNN creates scatterplots which are less fuzzy than NNP, being very close to the ones that t-SNE creates. For test data, note that both NNP and KNNP place point clusters at different locations than t-SNE. This is expected since, as explained, t-SNE is non-parametric. We also see that KNNP delivers visually identical plots for approximate *vs* exact search. Hence, we can use the faster approximate (ANN) search without fear of quality loss.

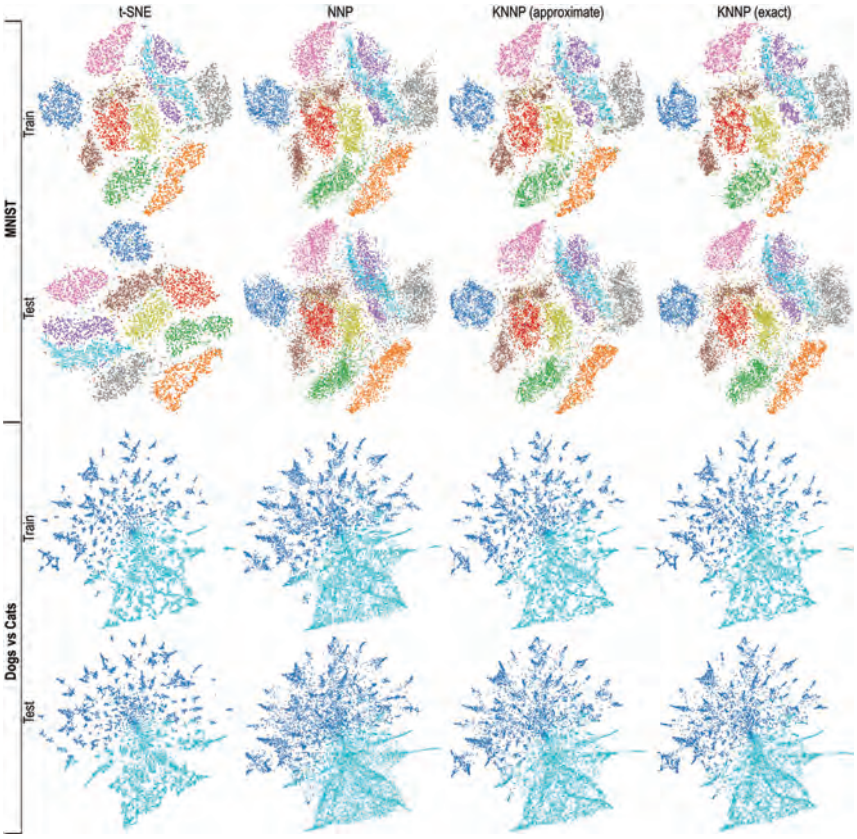


Figure 6.15: Projections created by t-SNE, KNN, and KNNP (approximate and exact search variants) on MNIST and Dogs *vs* Cats datasets during training and testing (inference).

6.8 DISCUSSION AND CONCLUSIONS

We presented an in-depth study aimed at assessing and improving the quality of dimensionality reduction (DR) using supervised deep learning. For this, we first explored the design space of NNP in six directions: training-set size, network architecture, regularization, optimization, data augmentation, and loss functions. We sampled each direction using several settings (method types and parameter values) and compared the resulting projections with the ground-truth (t-SNE method) quantitatively, using four quality metrics, and also qualitatively by visual inspection. Our exploration delivered an optimal hyperparameter setting that brings NNP closer to the quality of the t-SNE ground truth. Separately, we showed that NNP is stable with respect to all parameter settings, training-set size, and noise added to the input data.

Secondly, we further improved NNP quality by proposing KNNP, a refinement of the method that learns by projecting entire neighborhoods rather than individual samples. While improving quality, KNNP keeps the same attractive features of NNP, namely computational scalability, out-of-sample capability, and robustness to parameter settings. We also inferred optimal parameter settings for KNNP ($K = 5$ nearest neighbors found by

approximate fast search) apart from the already established parameters it inherits from NNP discussed above.

Our results complement recent evaluations [52, 58] and show that supervised deep learning is a practical, robust, simple-to-set-up, and high-quality alternative to t-SNE for dimensionality reduction in data visualization. More broadly, we believe that our methodology can be directly used to reach the same goals (optimal settings and proof of stability) for any projection technique under study, whether using deep learning or not. We plan to extend these results in several directions. First, we aim to generalize the (K)NNP approach to work without the need of supervision. Secondly, we aim to extend our approach to project time-dependent data in a stable and out-of-sample manner, which is a long-standing, but not yet reached, goal for high-dimensional data visualization.

In Chapter 7, we present NNInv, a technique inspired by NNP and KNNP that is used to create *inverse* projections, *i.e.*, the maping from low- to high-dimensional space. We present several experiments that show its quality over different datasets and when compared to other inverse projection techniques.

7 DEEP LEARNING INVERSE PROJECTIONS

7.1 INTRODUCTION

As we already explained, multidimensional projections take observations from a high-dimensional dataset D and generate a typically two-dimensional scatterplot $P(D)$. The usefulness of those projections is detailed in these previous chapters, as well as the various applications of projections in data science and information visualization at large. *Inverse projections* aim at the opposite task: given some dataset D and its 2D projection $P(D)$, inverse projections create a mapping between the two spaces, which can be used to infer a high-dimensional point $\mathbf{x} \in \mathbb{R}^N$ (not necessarily part of D) based on *any* point $\mathbf{y} \in \mathbb{R}^2$ (not necessarily part of $P(D)$). This enables the user to *generate* points that are between existing high-dimensional points by selecting regions in 2D space.

Computing inverse projections has received increased importance in visual analytics and beyond. The iLAMP [5] and Radial Basis Functions (RBF) [6] inverse projections were used to interpolate between high-dimensional observations so that these vary smoothly in a given projection, for 3D shape morphing applications. The same methods were adapted to create space-filling dense maps of decision zones of classifiers to help understanding their behavior in machine learning engineering [58, 171] (Chapter 8). Dense maps are prominently featured in the well-known TensorFlow framework for classifier engineering [191]. However, computing inverse projections is hard: Algorithms like iLAMP and RBF are slow, have multiple free parameters, and their quality strongly depends on the dataset D and direct projection technique P being used [58]. Moreover, compared to the tens of direct projection algorithms discussed in Chapter 3, only a handful – to our knowledge, only the iLAMP and RBF techniques listed above for the general case – inverse projection techniques exist.

In Chapter 5, we described NNP, a technique to deep-learn *any* direct projection technique, and showed that this approach leads to high-quality projections, among other desirable features. The question then rises naturally: Can we use a similar deep-learning approach to construct inverse projections? This is precisely what NNInv, the technique presented in this chapter, does. Given a dataset D and projection $P(D)$, computed by any desired, user-chosen, technique P, we *learn* the inverse mapping $P(D) \rightarrow D$ using a deep neural network. Next, we use the learned mapping to project any sample $\mathbf{y} \in \mathbb{R}^2$ to nD space. We validate the accuracy, speed, and ease of use of our technique using both quantitative quality metrics and dense maps on a couple of real-world datasets, and compare our results with other techniques such as iLAMP and RBF.

This chapter is structured as follows. Section 7.2 details our method. Section 7.3 presents our results. Section 7.4 concludes the chapter.

7.2 METHOD

We start with a dataset $D \subset \mathbb{R}^N$ and a projection technique P. Both can be freely chosen by users depending *e.g.* on their application of interest and the features that P should

This chapter is based on publication [59].

manifest, *e.g.*, good cluster segregation, distance preservation, or any other known quality metrics, as described in Chapters 2 and 3. We hypothesize that the way in which P captures the data structure in D can be used to create an inverse projection P^{-1} by using a small training set $D_s \subset D$ and its respective projection $P(D_s) \subset P(D)$. We next construct P^{-1} by training a neural network on the training set $T_s = (D_s, P(D_s))$, with D_s selected by random sampling of D. We use the remaining data $T_p = (D \backslash D_s, P(D) \backslash P(D_s))$, unseen during training, for validation. The cost function aims to generate samples in D that are as close as possible to the training ones in D_s. Summarizing, our method has three steps: In step 1, we create the projection $P(D_s)$ of the training samples D_s using any desired projection technique P. In step 2, we train a neural network using the training set T_s. In step 3, we validate the trained network using the test set T_p. The trained network is our inverse projection P^{-1}. For any given 2D point $\mathbf{y}$, we can now infer its high-dimensional counterpart by $P^{-1}(\mathbf{y})$.

After extensive empirical testing, varying the number of layers, units per layer, and activation functions, we set the architecture of P^{-1} to four fully-connected hidden layers, with 2048 units each, using ReLU activation functions, followed by an n-element layer, which uses a sigmoid activation to encode the inverse projection, scaled to the interval $[0, 1]$ for implementation simplicity – that is, we assume that our high-dimensional data resides in $[0, 1]^N$ instead of $\mathbb{R}^N$. We initialize weights with the He uniform-variance scaling initializer [82], and bias elements by a constant value 0.01, which showed good results during testing. We use the Adam [102] optimizer to *train* P^{-1} for up to 300 epochs. We stop training automatically on convergence, defined as the moment when the validation loss stops decreasing. In practice, we need 150 epochs on average for convergence (see Section 7.3.1). As cost function, we use mean squared error, which showed better convergence speed during testing than mean absolute error and log hyperbolic cosine (logcosh). To test quality, we compare the nD inferred samples $P^{-1}(D_p)$ with ground truth D_p using the mean squared error metric.

7.3 RESULTS

We tested our method on the following:

Projections: We use for P t-SNE [127] and UMAP [138], which have high-quality and are well known in the dimensionality reduction community [148]. We also tested other methods such as PCA and LAMP [95], with similar results, omitted here for brevity.

Inverse projections: We compare our method with two alternatives: iLAMP [5] and RBF [6]). Besides PCA, these are the only inverse projection methods we are aware of. PCA shows poor results as both direct and inverse projections for data of high intrinisic dimensionality, so we omit this from the presentation.

Datasets: We use one synthetic dataset and two well-known real-world benchmark datasets in machine learning. The synthetic dataset (*Blobs*) has 60K observations sampled from a Gaussian distribution with 5 different centers (clusters) and 50 dimensions. For the real-world datasets we used MNIST and FashionMNIST, which are described in detail in Section 2.3.

We next discuss our method in terms of scalability (Section 7.3.1), quantitative assessment of quality (Section 7.3.2), and qualitative assessment of quality (Section 7.3.3).

7.3.1 *Scalability in training and inference*

Scalability implies the effort required to *train* our method and, separately, the effort needed to *infer* $P^{-1}(Y)$ as function of the size $|Y|$ of the dataset Y to inversely project. Table 16 shows the number of training epochs needed to obtain convergence (defined as in Section 7.2) as function of the training set size $|D_s|$, for all three considered datasets and P = t-SNE. The figures for other projections (UMAP, PCA) are very similar. Columns 2..4 indicate averages for multiple runs that select D_s by randomly sampling D (see Section 7.2). Overall, we see that we obtain convergence for roughly 150 epochs for all datasets and training-set sizes, and also that this number of epochs is quite stable for training-set sizes $|D_s|$ larger than 1K samples.

Table 16: Training effort until convergence.

	Average # epochs for each dataset D			
Training set size $\|D_s\|$	**Blobs**	**Fashion-MNIST**	**MNIST**	**Avg.**
500	268.0	214.0	213.5	192.5
1000	190.5	129.0	147.5	149.0
2000	153.0	112.0	111.0	112.5
5000	103.0	120.5	138.0	127.5
7000	127.0	118.5	151.0	144.0
10000	82.0	124.5	142.5	146.5
average $\|D_s\|$ per D	153.9	136.4	150.6	145.3

Figure 7.1 shows the inference speed for all three datasets. Note that speed does not depend on the projection method P, by construction. Also, in this experiment, we consider *any* point $\mathbf{y} \in \mathbb{R}^2$, *i.e.*, not only points in the test-set D_s, since we don't need ground truth information to assess speed, and since in actual use one would not have such ground truth available. We see that both RBF and iLAMP have a superlinear behavior, while NNInv (our method) is almost linear. More importantly, NNInv is roughly one magnitude order faster than RBF and nearly two orders of magnitude faster than iLAMP for 40K samples or more. This speed-up is crucial for applications that need to inversely project hundreds of thousands of samples (or more), like in the construction of dense maps (see [58, 171] and Section 7.3.3 next). In such cases, NNInv allows constructing such maps in seconds, whereas iLAMP and RBF require (tens of) minutes, which makes human-in-the-loop usage of such dense maps impossible in visual analytics scenarios – which are one of the key reasons why dense maps are built in the first place.

7.3.2 *Quantitative Assessment of Quality*

Besides being fast, we want an inverse projection to be *accurate*. That is, given some ground truth pair ($\mathbf{x} \in \mathbb{R}^n, \mathbf{y} = P(\mathbf{x}) \in \mathbb{R}^2$), *unseen* by training, we want that $P^{-1}(\mathbf{y})$ be as close as possible to $\mathbf{x}$. This follows the same idea as, on the one hand, normalized stress metrics used to gauge the quality of projections in the literature [128, 192], and on the other hand classical validation of inference models in machine learning. We measure quality in our case by computing the average inverse-projection mean square error $MSE = \|\mathbf{x} - P^{-1}(P(\mathbf{x}))\|^2/|D_p|$ over the test set D_p. The closer MSE is to zero, the better P^{-1} is. Figure 7.2 shows MSE for our three datasets, two projections (t-SNE and UMAP), three tested inverse projections (iLAMP, RBF, and NNInv). We also consider several training-set sizes $|D_s|$ to show how MSE depends on the training amount. For *Blobs*, a relatively easy-to-project synthetic data, all methods have basically zero error, except

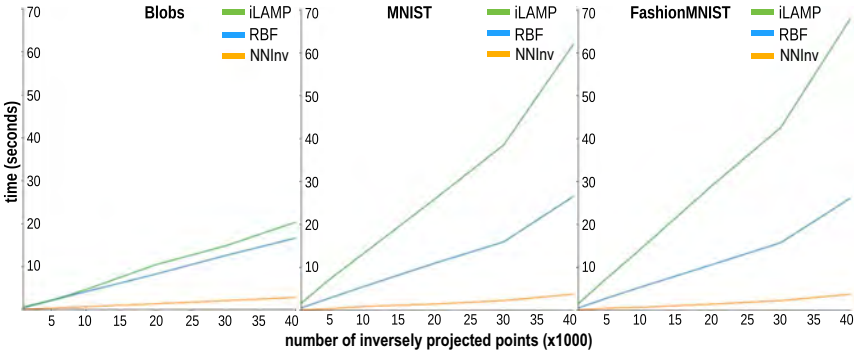


Figure 7.1: Inverse projection speed as function of number of samples.

RBF. *MNIST* and *FashionMNIST* show similar behavior: Our method (NNInv) achieves consistently lowest error. The second-best method is iLAMP. Errors are larger for these real-world complex datasets than for the synthetic *Blob*, which is expected.

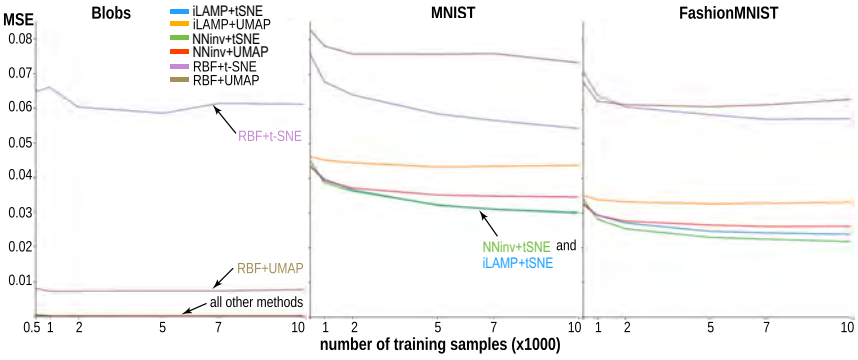


Figure 7.2: Mean square error of inverse projection (lower=better).

7.3.3 *Qualitative Assessment of Quality*

We now show why having a fast and accurate inverse projection is important for a concrete application – understanding the decision zones of classifiers. In supervised machine learning, a classifier M_c is trained on a *labeled* training-set D_s, which typically sparsely samples some continuous universe U ($U \subset \mathbb{R}^n$ in our case), and partitions U in so-called *decision zones* Z_i, so that any $\mathbf{x} \in Z_i$ gets label l_i at inference time. The classifier accuracy is evaluated using a labeled test-set D_p. Visualizing the decision zones Z_i helps understanding the behavior of M_c, as well as areas in U where training may have been suboptimal and thus may need extra effort. Recently, a method to construct dense decision maps using inverse projections was proposed [171]. Briefly put, every pixel $\mathbf{y}$ in an image is colored by the label assigned to $P^{-1}(\mathbf{y})$ by a classifier M_c trained on the same data used to construct the projection P from which P^{-1} was derived. The quality of such decision maps depends on the projection P used, with t-SNE and UMAP performing better than several other tested methods [58]. However, while acknowledging its impor-

tance, the effect on quality of the *inverse* projection P^{-1} used was not tested. We alleviate the above limitation in [58] and also show why our inverse projection achieves better results for dense maps visualizing classifier decision zones. For this, we construct decision maps for projections $P \in \{\text{tSNE}, \text{UMAP}\}$, datasets $D \in \{\text{Blobs}, \text{MNIST}, \text{FashionMNIST}\}$, inverse projections $P^{-1} \in \{\text{iLAMP}, \text{RBFp}, \text{RBFc}, \text{NNInv}\}$, and classifiers $C \in \{\text{LR}, \text{CNN}\}$. Here, RBFp and RBFc are two versions of the RBF inverse projection, using fixed control points, respectively control points defined as centers of clusters obtained from the input data D (for details, we refer to the original paper [6]). *LR* is a simple logistic regression classifier, used since we know it produces piecewise-linear decision boundaries and hyperpolyhedral decision zones; and CNN is a convolutional neural network, which we know it works well for image data like (Fashion)MNIST. All decision maps are images of 500^2 pixels, so $|D_p| = 250000$ points (Fig. 7.3). Importantly, all maps were constructed completely from *unseen* data – that is, we do not use any of the data points or their projections present in the training set D_s. We discuss our results next.

Blobs dataset: As expected, for this simple dataset, both t-SNE and UMAP separate well the 5 clusters present in the data. The LR trained on this dataset achieved 100% accuracy. All inverse projections P^{-1} appear as compact zones that surround the corresponding projection scatterplots. For the LR classifier, we *know* that the decision boundaries should be piecewise linear. UMAP yields more concentrated clusters, so the corresponding dense maps resemble very much Voronoi diagrams of the respective cluster configurations – which is indeed expected, and a positive sign of the correctness of the dense maps. For the t-SNE projection, iLAMP and NNInv are closest to such linear boundaries, while RBFp and RBFc create more jagged boundaries. This is a first hint that iLAMP and NNInv are better inverse projections.

MNIST dataset: The CNN classifier used obtained a 99.6% training-set accuracy. As the projection (and underlying dataset) is more complex, the inverse projections are more challenged. Recent studies have empirically shown that decision zones of such neural networks, used for natural-image dataset classification, are *connected*, with relatively *smooth* boundaries [64]. Hence, we *expect* our dense maps to show this. In Fig. 7.3, we first observe that both iLAMP and NNInv are closest to the above properties, while RBFc generates highly noisy, sprayed-points-like, disconnected, and complex-shaped decision zones (see dashed-line annotations in figure). These generate the false impression that the classifier has difficulties for such samples, which is not true, given the observed accuracy. RBFp also generates noisy/disconnected zones, albeit less than RBFc, but more than iLAMP and NNInv. Both RBFp and RBFc also generate visible 'false islands', *i.e.*, significant-size areas in the decision maps that have a label which does not match any significant number of points having the same label in the scatterplots (see continuous-line annotations in figure). These convey the false impression that the classifier creates certain decision zones in areas where actually nothing like this happens. While both above phenomena exist also for NNInv, this is to far smaller extents.

FashionMNIST dataset: The CNN classifier used obtained a 98.7% training-set accuracy. We can make the same observations made for MNIST's decision zones, even to stronger extents. RBFc and RBFp generate highly fragmented, jagged, and disconnected decision zones, with RBFp being better than RBFc. iLAMP and NNInv generate smoother, more connected, and quite similar zones. This is quite interesting, since the two methods are completely different. However, iLAMP generates noisier zones and more jagged

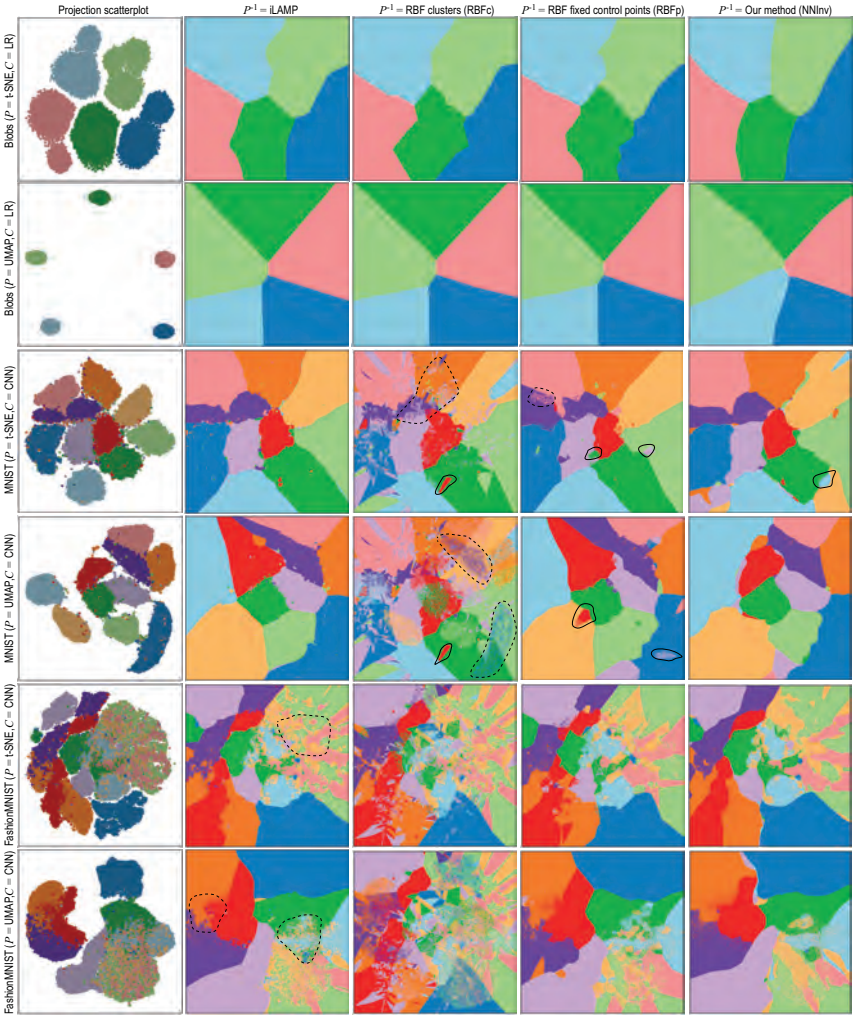


Figure 7.3: Dense maps constructed for combinations of classifiers M_c, projections P, inverse projections P^{-1}, and datasets. See Section 7.3.3.

boundaries (see annotations in figure). Given, again, the mentioned insights on how such zones/boundaries should be [64], we find NNInv being better than iLAMP.

7.4 DISCUSSION AND CONCLUSION

In this chapter, we presented NNInv, a new method for computing inverse projections from 2D to high-dimensional data spaces by learning the behavior of a direct projection method. Our method is generic (can handle any direct projection method and type of high-dimensional dataset), easy to use (does not require any user parameters), one to two orders of magnitude faster than existing inverse projection methods, and simple to implement using existing out-of-the-box deep learning toolkits [34]. We compared our method on three datasets, two state-of-the-art projections (UMAP and t-SNE), against three inverse projection methods (iLAMP, RBFc, and RBFp). We found our method to deliver higher accuracy, and decision zones that match equally well or better to known properties of such zones for both simple (linear regression) and more complex (convolutional neural network) classifiers.

Our method can be extended in several directions. First, the design space of its underlying neural network can be better explored to reach higher accuracy and/or less training effort. Secondly, different quality metrics can be used to deliver inverse projections which are specifically suited for specialized tasks such as assessing confusion zones of classifiers. Finally, we can apply our inverse projection to support more applications beyond decision map exploration in machine learning.

In Chapters 8 and 9, we present two applications of direct and inverse projections, namely, a technique for drawing machine learning classifier decision boundaries, and a technique for drawing multivariate functions, in the context of optimization problems.

8 CLASSIFIER DECISION BOUNDARY MAPS

8.1 INTRODUCTION

Advances in machine learning (ML) enabled breakthroughs in application areas such as computer vision, image processing, path planning and business intelligence. However, most ML methods still work largely as *black boxes*, due to the lack of interpretability behind the decision functions they employ. As such, methods can become very complex, as in the case of deep learning (DL) methods, practitioners and users have challenges in understanding, customizing and trusting them [169, 197]. To alleviate this, recent work has focused on visually explaining how ML techniques learn and take their decisions [41, 70, 167, 168].

One such interpretability challenge regards the so-called *decision boundaries* of classifiers. Formally put, let D be the sample space input for a classifier. The classifier can be seen as a function f that assigns a class label to every point in D. Understanding how f, defined by the training process, partitions D into same-class *regions*, separated by decision *boundaries*, can help many tasks related to classifier design, for example, locate how training samples affect the classification of test samples close to them in D; spot areas in D that require more or have redundant training samples; and find if the classifier technique used is too "stiff" to separate complex labeled sample distributions in D [81, 141].

Visualizing complex-shaped decision boundaries embedded in a high-dimensional space D is very challenging. All existing solutions essentially project D to $\mathbb{R}^2$ by some projection method P so as to visualize the boundaries and/or zones. A recent method in this area [171] proposes an improvement with respect to the classical way of visualizing projections as color-coded sparse scatterplots, by creating *dense maps* where every 2D image pixel encodes one or more high-dimensional points and their assigned labels. This way, the user effectively sees how the classifier partitions the high-dimensional space into decision zones having different labels. An earlier similar technique was proposed [180]. In contrast to that work, we propose here to evaluate and extend the method proposed in [171], due to its simplicity and genericity.

The quality of dense maps constructed crucially depends on two factors: (a) the quality of the projection P and (b) the quality of the inverse projection P^{-1}. A recent study [58] addressed the first by studying a set of 28 projection techniques on 4 classifiers and 2 real-world datasets and outlining good combinations. Separately, aspect (b) was addressed by proposing a new inverse projection technique called NNInv (Chapter 7), which is faster and more accurate than state-of-the-art ones.

Although addressing computational challenges, both these recent developments [58, 59] do not further elaborate on the *interpretability* of decision maps. Specifically, even for good-quality direct and inverse projections, such maps can still show significant noise, visible as jagged boundaries and/or isolated salt-and-pepper islands, which are due to inherent distortions caused when mapping this high-dimensional space D to 2D.

This chapter is based on publication [170]

Such artifacts can further influence the way that users interpret the behavior of the studied classifiers.

In this chapter, we extend previous work [58] by proposing several new techniques that address the above points. First, we propose a quality metric to gauge the correctness of the direct projection and use it to selectively remove badly-projected points, while keeping the overall structure of the projection intact. This decreases the amount of noise along decision boundaries and also removes spurious small-scale islands. Secondly, we propose several ways to visualize the high-dimensional distances between samples in the 2D projection. This lets users see which areas in the data space are close(st) to decision boundaries, thus, potentially important to further study or refine, for example, by data augmentation. To obtain the best quality, we combine in our work the direct projections suggested previously [58] with the novel NNInv inverse projection.

The structure of this chapter is as follows. Section 8.2 overviews related work. Section 8.3 presents the experimental setup used to study how dense maps depend on DR techniques and classifiers, covering a combination of 28 DR techniques and 4 classifiers. Section 8.4 presents and discusses our results regarding the choice of DR techniques. Section 8.5 introduces our new filtering technique for removing artifacts in dense maps caused by projection distortions. Section 8.6 introduces a new technique for highlighting data-space points close to decision boundaries based on the computation and rendering of the high-dimensional distance-to-boundary. Section 8.8 concludes the chapter.

8.2 BACKGROUND

8.2.1 *Preliminaries*

As outlined in Section 8.1, S is sampled from a particular universe or subspace, $D \subset \mathbb{R}^N$, for example,the space of all images of digits [112]. A *classifier* for D is a function $f : D \rightarrow C$ which associates to every $\mathbf{x} \in D$ a class label from a categorical domain M_c, for example, the digits 0 to 9. The function f is constructed via a training set $S_t = \{(\mathbf{x}_i, c_i) | \mathbf{x}_i \in D, c_i \in C\}$ and tested via a similar but disjoint test set S_T. Different machine learning (ML) techniques exist to construct f, some of the best known being k-Nearest Neighbors (k-NN), Logistic Regression (LR), Support Vector Machines (SVM) [39], Random Forests (RF) [26] and Convolutional Neural Networks (CNN) [106]. The *decision zone* for a label $c \in C$ is then the set $DZ(c) = \{\mathbf{x} \in D | f(\mathbf{x}) = c\}$, with *boundary* $\partial DZ(c)$. Such sets are typically compact, given the underlying contiguity hypothesis usual in many ML contexts [132].

8.2.2 *Decision Boundary Maps*

Exploring how well f was learned from S_t is typically done by comparing how well the inferred labels $f(\mathbf{x}_i)$ match the actual labels c_i for all $\mathbf{x}_i \in S_T$. To visualize these, one typically constructs a scatterplot $P(\mathbf{x}) | \mathbf{x} \in S_T$ of such points, color-coded by their labels. The underlying idea is that, if P preserves data structure and assuming a relatively smooth behavior of the classifier f, then decision zones will appear as same-color point clusters in the scatterplot. Conversely, differently colored "outlier" points in a cluster typically indicate classification problems. While simple to construct, such scatterplots do not show how the classifier f labels the *entire* universe D but only a sparse sampling S_T thereof. Simply put, we do not know what happens in the blank space *between* the scatterplot points. In particular, the decision boundaries $\partial DZ(c)$ of the classifier are not explicitly visualized, leaving the user to guessing their actual position [168].

Image-based *dense maps* improve upon this by coloring each pixel of the target (screen) image by the assigned label(s) of samples in D that project there [81, 141]. Recently, the Decision Boundary Maps [171] technique was proposed (see also Figure 8.1a): For every pixel $\mathbf{y}$ of the target (projection) space, data samples $\mathbf{x} \in D$ are created, by gathering the scatterplot points $Y = \{P(\mathbf{x})\}$ that project into $\mathbf{y}$. If this yields fewer than U points, one adds to Y $U - |Y|$ synthetically created points $P^{-1}(\mathbf{y}')$, where $\mathbf{y}'$ are random points in the pixel $\mathbf{y}$. Finally, the respective pixel is colored to reflect the labels $f(\mathbf{x} \in Y)$ with hue mapping label value and saturation label variation over Y, respectively.

Decision boundary maps are independent on the classifier technique f being studied; have no complex-to-set free parameters; and effectively create dense maps where each image pixel is colored to reflect how f behaves for the nD point(s) that project there via P. Decision zones $DZ(c)$ are directly visible as all image pixels showing M_c's color and decision boundaries $\partial DZ(c)$ show up as pixels having different-color neighbors. Few compact zones with simple (smooth) boundaries tell that the classifier has little difficulty in taking decisions over D. Multiple disjoint same-color zones and/or zones with tortuous boundaries tell the opposite. Small-size "islands" of one color embedded in large zones of different colors suggest misclassifications and/or training problems.

To compute decision maps, t-SNE [127] and LAMP [95] were used to implement P and iLAMP [5] was used for P^{-1}, respectively. More recently, NNInv was proposed, which is a more accurate and faster to compute implementation for P^{-1}, based on deep learning [59]. It is worth noting that both NNinv and iLAMP *fit* P^{-1} from data. That is, given a set of projected points $Y \subset \mathbb{R}^2$ and their nD counterparts $X \subset \mathbb{R}^N$, these methods learn a mapping $P^{-1} : \mathbb{R}^2 \rightarrow \mathbb{R}^N$ through an optimization process that aims to minimize a reconstruction error. The difference is in what and how is optimized for: While iLAMP generates inverse samples by returning a weighted average of nD data points based on the distances between their 2D counterparts, NNInv fits by regression a function that better suits the training data using Neural Networks. As P^{-1} is *inferred* from finite data, it can be used to invert *any* DR method. Hence, iLAMP and NNinv are especially useful for projection methods that do not provide an inverse mapping, for example, t-SNE. However, if desired, they can also be used for projection methods that do provide an inverse function, for example, Principal Component Analysis (PCA). A separate question is which is a suitable implementation for the direct projection P, since it is well known that different DR methods create widely different projections for the same input [56, 128, 148, 192].

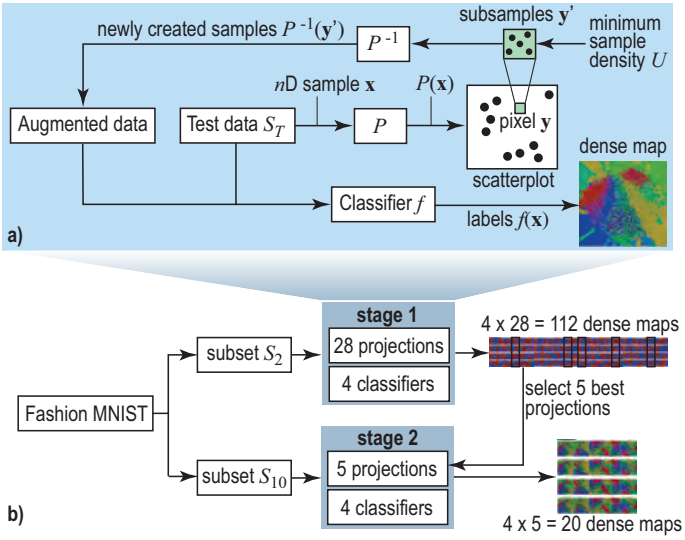


Figure 8.1: (**a**) Dense map construction algorithm; (**b**) Two-phase experiment set-up.

8.3 EXPERIMENT SETUP

As explained in Section 8.2.2, the quality of a decision map depends heavily on the choice of projection technique P used. Consider, for example, a toy two-class k-NN classifier for a 3D data space $D \subset \mathbb{R}^3$ trained with a simple S_t consisting of one sample of each class. We know in this case that the decision boundary should be a plane halfway the two training samples. So, a good 2D projection P should ideally render two compact half-planar decision zones. Conversely, a poor P may create several same-class zones having complex curved boundaries; if we saw such an image, we would wrongly judge the behavior of this classifier.

To study which of the many projection techniques in existence are most suitable for constructing effective decision maps, we designed and executed a two-stage experiment, as follows (see Figure 8.1b).

Data: We select two different subsets of the Fashion MNIST dataset, which is described in detail in Section 2.3, as follows:

- S_2: A two-class subset (classes *T-Shirt* and *Ankle Boot*) that we hand-picked to be linearly-separable;
- S_{10}: An all-class subset (*T-Shirt*, *Trouser*, *Pullover*, *Dress*, *Coat*, *Sandal*, *Shirt*, *Sneaker*, *Bag* and *Ankle Boot*). This is a non-linearly-separable dataset.

Classifiers: We consider the same classifiers used in the original decision boudary map paper [171]: LR, RF, k-NN and CNN. Please refer to Table 32 in Section A.4 for implementation details. For CNN, we used two convolutional layers with 64 filters each and 3×3 kernels, followed by one 4096-element fully-connected layer, trained with the Adam optimizer [102]. These classifiers create very different decision boundaries: At one extreme, LR boundaries are linear (hyperplanes). k-NN boundaries are piecewise-linear (facets of nD convex polyhedra). RF creates typically more complex boundaries than

k-NN. At the other extreme, CNN boundaries can have arbitrarily complex topologies and geometries, due to the complex decision function f coded by the deep network structure. However, CNNs are known to perform very well for classifying images like our dataset, while at the other extreme simple classifiers like LR are highly challenged by such data.

Training: The four classifiers were separately trained on the two subsets S_2 ($|S_t| = 2160$ samples, $|S_T| = 240$ samples) and S_{10} ($|S_t| = 10,800$ samples, $|S_T| = 1200$ samples). We verified that the training yielded good accuracies in all cases (Table 17). This is essential to know when we next gauge the dense maps' ability to capture a classifier behavior (see stage 1 below).

Table 17: Accuracy of classifiers, 2-class and 10-class problems.

Classifier Technique	**2-Class**	**10-Class**
Logistic Regression (LR)	1.0000	
Random Forest (RF)	1.0000	0.8332
k-Nearest Neighbors (KNN)	0.9992	0.8613
Convolutional Neural Network (CNN)	1.0000	0.9080

Projections: Table 18 lists the 28 selected projection techniques (P) to create dense maps as well as the parameter settings (default indicates using the standard ones the algorithms come with). As selection criteria, we considered well-known projections of high quality, as per the evaluation in Chapter 3, good computational scalability, ease of use (P should come with well-documented parameter presets) and publicly available implementation.

Table 18: Projections tested in phase 1 (Section 8.4.2). Projections tested in phase 2 (Section 8.4.2) are marked in bold.

Projection	**Parameters**
Factor Analysis [97]	iter: 1000
Fast Independent Component Analysis (FastICA) [93]	fun: exp, iter: 200
Fastmap [63]	default parameters
IDMAP [142]	default parameters
Isomap [203]	neighbors: 7, iter: 100
Kernel PCA (Linear) [181]	default parameters
Kernel PCA (Polynomial)	degree: 2
Kernel PCA (RBF)	default parameters
Kernel PCA (Sigmoid)	default parameters
Local Affine Multidimensional Projection (LAMP) [95]	iter: 100, delta: 8.0
Landmark Isomap [33]	neighbors: 8
Laplacian Eigenmaps [17]	default parameters
Local Linear Embedding (LLE) [175]	neighbors: 7, iter: 100
LLE (Hessian) [46]	neighbors: 7, iter: 100
LLE (Modified) [229]	neighbors: 7, iter: 100
Local tangent space alignment (LTSA) [230]	neighbors: 7, iter: 100
Multidimensional Scaling (MDS) (Metric) [109]	init: 4, iter: 300
MDS (Non-Metric)	init: 4, iter: 300
Principal Component Analysis (PCA) [97]	default parameters
Part-Linear Multidimensional Projection (PLMP) [155]	default parameters
Piecewise Least-Square Projection (PLSP) [155]	default parameters
Projection By Clustering [153]	default parameters
Random Projection (Gaussian) [43]	default parameters
Random Projection (Sparse) [43]	default parameters
Rapid Sammon [156]	default parameters
Sparse PCA [231]	iter: 1000
t-Stochastic Neighbor Embedding (t-SNE) [127]	perplexity: 20, iter: 3000
Uniform Manifold Approximation (UMAP) [138]	neighbors: 10

Dense maps: We use a two-stage creation and analysis of dense maps, as follows (Figure 8.1b). In stage 1, for S_2, we create dense maps using all 28 projections for all 4 classifiers, yielding a total of 112 dense maps. All maps have a 400×400 pixel resolution. Since S_2 is quite simple (two linearly separable classes), *and* since all classifiers for S_2 have very high accuracies (Table 17), the resulting maps should display (ideally) two compact zones separated by a smooth, ideally linear, boundary. We visually verify which of the 112 maps best comply with these criteria and next select the five projections (of the 28 tested ones) which realize these maps. These are shown in bold in Table 18. Next, in step 2 of the study, we create dense maps, for all 4 classifiers again but using the more complex S_{10} dataset. Finally, we explore these visually to gain fine-grained insights allowing us to further comment on the dense-map suitability of these 5 hand-picked projections.

8.4 ANALYSIS OF EVALUATION RESULTS

We next discuss the results and insights obtained in our two-stage experiment.

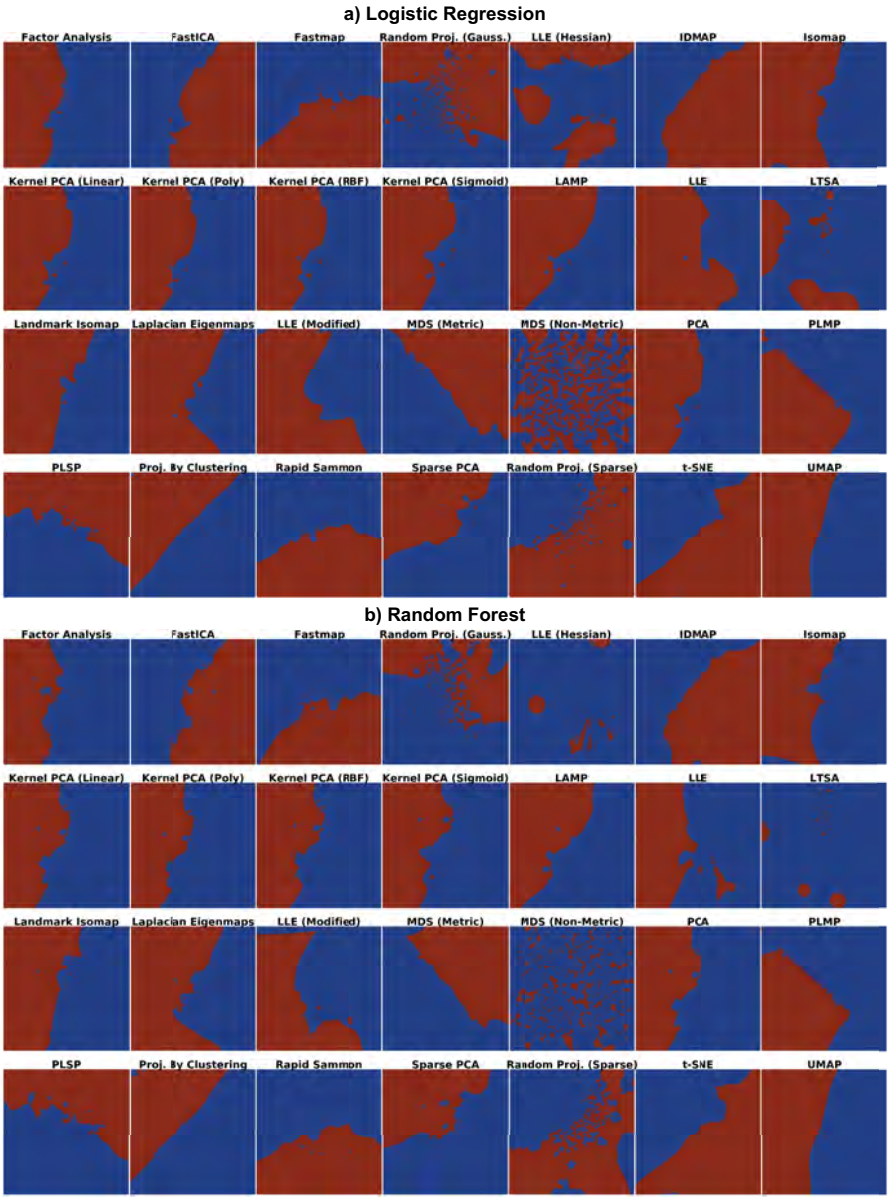


Figure 8.2: Dense maps for Logistic Regression (**a**) and Random Forest (**b**) classifiers on the 2-class S_2 dataset, all 28 tested projections.

8.4.1 *Phase 1: Picking the Best Projections*

As stated in Section 8.3, all four tested classifiers yield almost perfect accuracy for the simple 2-class problem S_2 (Table 17). Hence, their decision boundaries are "where they should be", that is, perfectly separating the two classes in S_2. Moreover, since S_2 is by

construction linearly separable, the dense maps constructed for these classifiers should clearly show two compact decision zones separated by a smooth, simple, boundary. We use this as a visual criterion to rank how well the tested projection techniques can achieve this. Figures 8.2 and 8.3 show the dense maps for all 28 tested projections versus the four tested classifiers, where red and blue indicate pixels mapping samples classified to one of the two labels in S_2. Interestingly, we see that even for this very simple problem not all projections perform the same. Our key observations are as follows:

Stability: The dense maps are surprisingly stable for the same projection over all four classifiers, except for LLE, LTSA, Random Projection (Gaussian) and Random Projection (Sparse). Hence, we already flag these four projections as less suitable.

Smoothness: All projections have relatively smooth boundaries, except Random Projection (Gaussian), Random Projection (Sparse) and MDS (Non-Metric). Since we expect smooth boundaries, these projections are less suitable. The projections which yield boundaries closest on average to the expected straight line are MDS, UMAP, Projection by Clustering, t-SNE and PLMP.

Compactness: Projections succeed up to widely different degrees in creating the expected two compact, genus-zero, decision zones. t-SNE, UMAP, Projection by Clustering and IDMAP do this almost perfectly. MDS (Non-Metric), the two Random Projections, LLE (Hessian) and LTSA perform the worst.

Summarizing the above, we select MDS (Metric), PLMP, Projection by Clustering, UMAP and t-SNE as the overall best projections to analyze further in phase 2, discussed next.

8.4.2 *Phase 2: Refined Insights on Data*

We now examine how the five projections selected in phase 1 perform on the 10-class dataset S_{10}, which is a harder classification problem. We already see this in the lower achieved accuracies (Table 17). Hence, we expect to have significantly more complex boundaries. Figure 8.4, that shows the dense maps for our 4 classifiers for the 5 selected projections, confirms this. Several interesting patterns are visible, as follows:

Overall comparison: For a given projection, the dense map patterns are quite similar over all four tested classifiers. This is correct, since the dense map is constructed based on the scatterplot created by that projection from the test set S_T, which is fixed. The variations seen along columns in Figure 8.4 are thus precisely those capturing the differences of decision boundaries of different classifiers. We see, for instance, that LR tends to create slightly simpler boundaries than the other three classifiers. Conversely, variations along rows in Figure 8.4 can be purely ascribed to the projection characteristics. Techniques designed to better separate data clusters, such as t-SNE and UMAP, show more compact decision zones with simpler boundaries than MDS, PLMP and Projection by Clustering. Also, the choice of neighborhood used internally by the projection technique to estimate points in the lower dimension (2D) does not seem to play a key influence: MDS, which uses global neighborhoods, shows similar pattern-variations along classifiers to the other four projections, all of which use local neighborhoods.

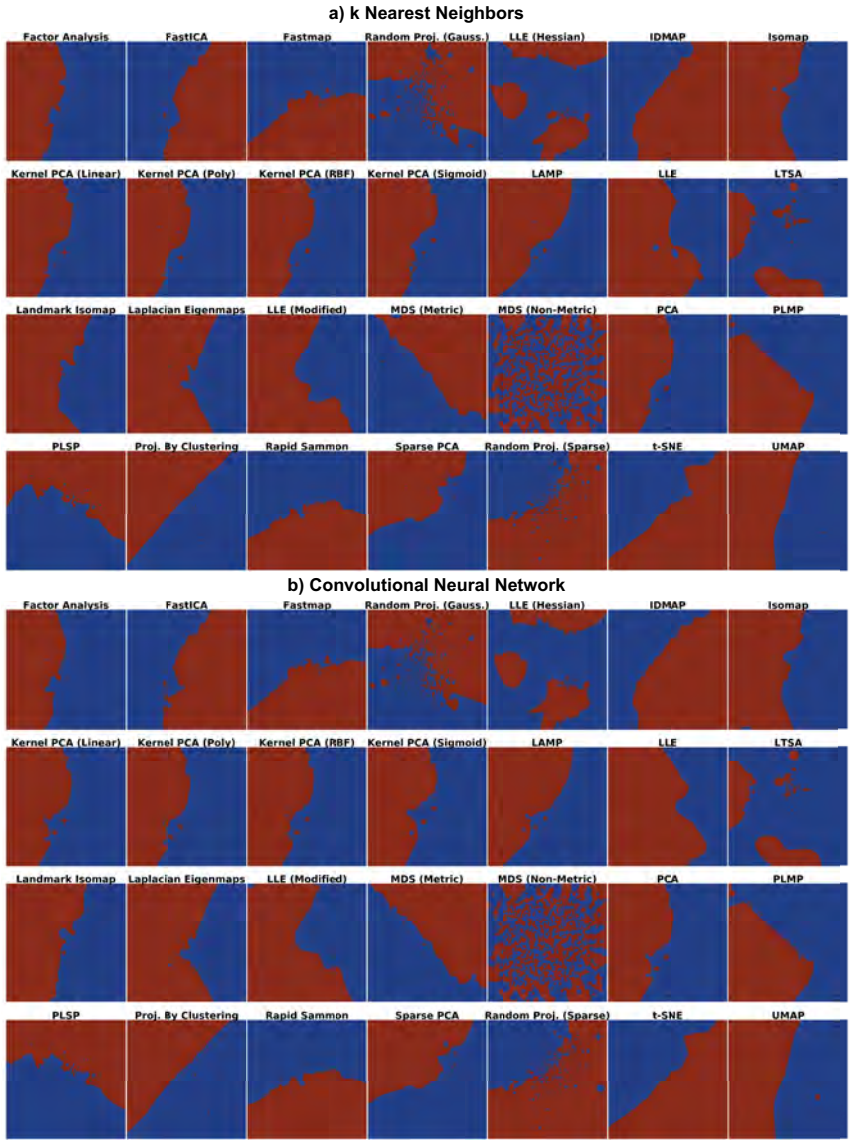


Figure 8.3: Dense maps for k-Nearest Neighbor (k-NN) (**a**) and Convolutional Neural Network (CNN) (**b**) classifiers on the 2-class S_2 dataset, all 28 tested projections.

Islands: The dense maps in Figure 8.4 show many small color *islands*. An island indicates that (at least) one sample was assigned a label different from the labels of samples that *project* close to it. In turn, this means that

(a) the island *does not* actually exist in the high-dimensional space D, so the projection P did a bad job in distance preservation when mapping nD points to 2D; or

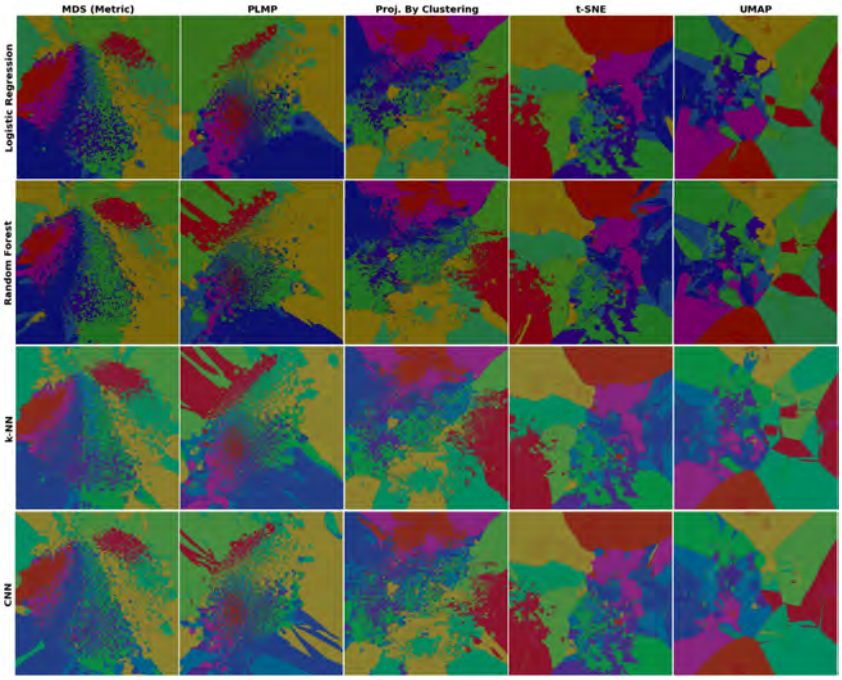


Figure 8.4: Dense maps for all classifiers, 10-class dataset, five best-performing projections.

(b) the island *may* exist in D, that is, there exist very similar samples that get assigned different labels. This case can be further split into

  (b1) the island *actually* exists in D, that is, similar points in D do indeed have different labels and the classifier did a good job capturing this; or

  (b2) the island *does not* exist in D, that is, the classifier misclassified points which are similar in the feature space but actually have different labels.

To understand which of these cases actually occur in Figure 8.4, we plot misclassified points atop the dense map as half-transparent white disks. Figure 8.5 shows this for the LR and CNN classifiers, all projections. Regions having many (densely packed) misclassifications show up as white areas. The insets (t-SNE dense map) exemplify how islands point to two of the above-mentioned issues: In Figure 8.4a, we see two very small color islands around the misclassified samples A and B. These islands indicate the extent up to which other samples, close to A or B, would also get misclassified. In contrast, the detail in Figure 8.4b shows a (red) island containing no white dots (misclassifications). This island either reflects a real variation of the label over similar points in D (case (b1) above) or else reflects a t-SNE projection artifact (case (a) above). To decide which of these cases actually occurs, we need additional techniques (discussed in Section 8.5).

Separately, we see that overall, the LR dense maps have more white dots than the CNN ones, which correlates with the lower LR accuracy (Table 17). We also see that the white points are *non-uniformly* spread over the dense maps by different projections. MDS and PLMP show many islands without white dots. As explained above, this either reflects densely-packed different-label points in D (case (b1)) or MDS and PLMP projection errors (case (a)). At the other extreme, t-SNE and even more so UMAP, strongly pack the

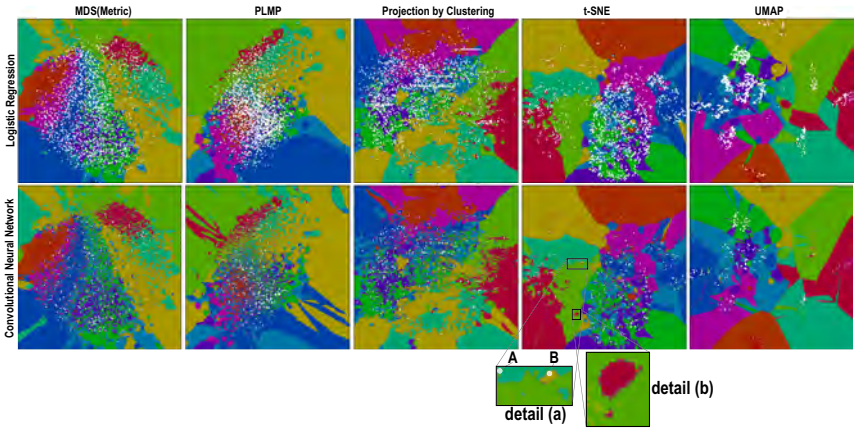


Figure 8.5: Classification errors (white dots) shown atop of the dense maps, logistic regression (LR) and CNN classifiers.

white dots, which tells that misclassifications actually occur for quite similar data samples. Densely-packed white points effectively show the *confusion zones*, so one can use them to decide which kinds of samples need to be further added to the training set to improve accuracy.

8.5 DENSE MAP FILTERING

As showed in Section 8.4.2, dense maps exhibit patterns such as non-smooth decision boundaries and/or small islands in the decision zones (Figures 8.4 and 8.5). As discussed there, such artifacts can be caused by either densely-packed different-label points in the data space D (case (b1)) or errors of the projection P (case (a)). For test data, for which we have ground-truth, we can disambiguate between these two cases—islands containing (many) misclassifications are likely due to case (b1), whereas the remaining islands are likely due to case (a).

However, using this method to interpret dense map images is suboptimal, since

- we need to interpret such maps also in actual inference mode (after testing), when no ground-truth labels are available;
- having to visually filter dense map artifacts like decision boundary jaggies and small islands is tedious.

Moreover, we note that such artifacts are very likely to happen *anyways*, even for a well-trained classifier (few misclassifications): Due to the ill-posed nature of DR, even the best performing projections P will eventually misplace points in a 2D scatterplot. This limitation is well known and discussed in several works [9, 56, 133, 134, 148]. The same limitations are shared by the inverse projection P^{-1} [5, 6, 59].

We propose to alleviate such artifacts by *filtering* the 2D scatterplot based on a quality metric that computes, locally, how well P preserves the high-dimensional data structure in D. Several such metrics exist, such as trustworthiness, continuity and normalized stress [56, 148]; neighborhood hit [95]; false neighbors, missing neighbors [133]; and the projection precision score [179]. Given our goals of characterizing how well a nD *compact* neighborhood maps to a similarly-compact 2D neighborhood, we use here the Jac-

card set-distance [134] between the k-nearest neighbors $v_k^2(i)$ of a point in the 2D projection and its neighbors $v_k^n(i)$ in nD, given by

$$JD_k(i) = \frac{|v_k^2(i) \cap v_k^n(i)|}{|v_k^2(i) \cup v_k^n(i)|}. \tag{8.1}$$

The JD value of a point i ranges between zero (if none of the 2D k-nearest neighbors of point i are among its nD k-nearest neighbors, worst case) and one (if all of its 2D k-nearest neighbors are exactly the same as its nD k-nearest neighbors, best case).

Having computed the JD rank (Equation (8.1)), we next filter out from the projection low-ranked points and construct the dense map from the remaining points as described in Section 8.2.2. Setting an absolute removal threshold is however hard and moreover depends on the neighborhood size k. To explain this, Figure 8.6 shows the distribution of number of samples per JD_k value for the MNIST dataset projected by t-SNE for four different k values. As visible, the distribution *shape* is relatively stable as function of k. As k increases, the distribution shifts to the right, as the likelihood that large neighborhoods coincide in 2D and nD increases—in the limit, when k equals to the total point count, $JD = 1$ for all points. Conversely, as k decreases, the distribution slightly shifts to the left, as the likelihood that neighbors of a point come in *exactly* the same order in 2D and nD is very small. Figure 8.6 shows a second, equally important, aspect, namely that the signal JD_k has a *discrete* nature. Indeed, for a given k, Equation (8.1) can take at most $k + 1$ different values.

Hence, for low k, JD_k splits the projected points in k bins, with relatively more points per bin as when using higher k values—compare for example, the vertical axes of the images in Figure 8.6 for low versus high k values. In turn, this means that setting an absolute threshold to eliminate low JD_k value points is hard: A too low threshold will eliminate too few points, while a slightly higher threshold may eliminate too many points. Hence, we proceed by (1) using a higher k value (roughly 10% of the dataset size) and next (2) we sort points on their JD_k value and remove the τ lowest-ranked points, where τ is a user-given percentage of the total dataset size.

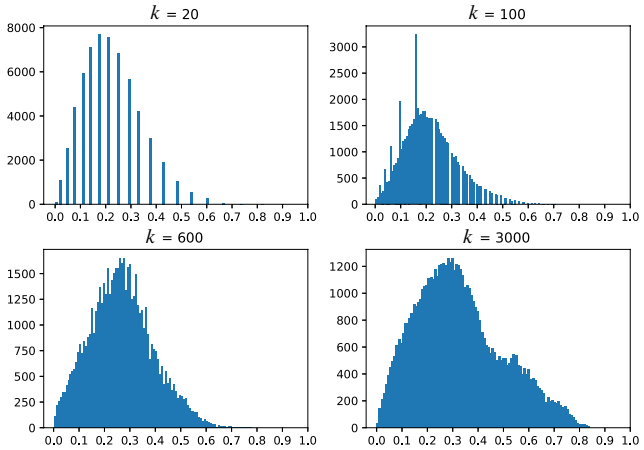


Figure 8.6: Histogram of JD_k rank for varying values of k for MNIST dataset, t-SNE projection.

Figure 8.7 shows results for different τ values for the MNIST dataset, projected by t-SNE. Setting τ is intuitive: Small values keep more data points, including potentially wrongly-projected ones, which cause islands and boundary jaggies in the dense maps. Larger values filter the projection better, yielding smoother decision boundaries and/or fewer islands due to projection problems but show fewer data in the final image. As visible, filtering does not change overall *size* and *shape* of the depicted decision zones, which is important, as it does not affect the insights that the filtered images convey. In practice, we found that τ values in the range of 15% to 20% of the dataset size give a good balance between removing artifacts and keeping enough data to have an insightful dense map. This is the setting used next in all images in this chapter.

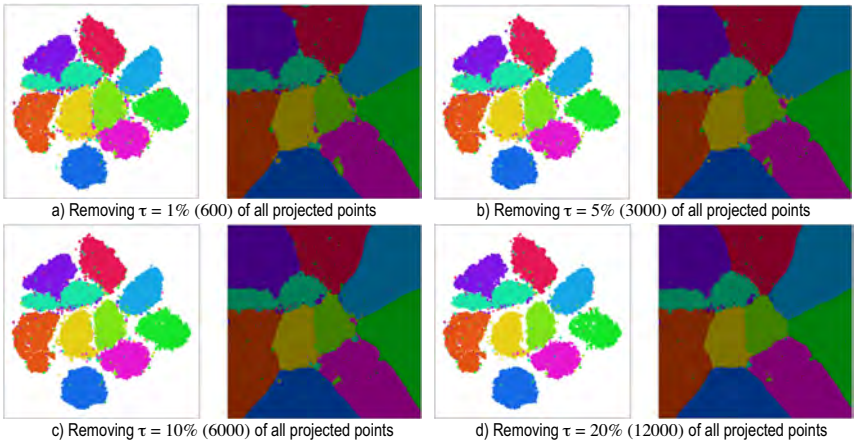


Figure 8.7: Removing poorly projected points with low JD_k ranks to filter dense map artifacts for the MNIST dataset, projected by t-SNE, inversely projected by iLAMP.

8.6 DISTANCE-ENRICHED DENSE MAPS

The dense map filtering effectively removes many of the confusing small-scale *islands* created by projection errors, thus, creates simpler-to-inspect decision zones. As already explained, a key use-case for these is for users to see which points (in the data space D) are close, respectively far away from, the decision boundaries. The distance-to-boundary information indicates the classification confidence—so, if a classifier performs poorly, one can use this distance to infer on what kind data in D such problems occur and next alleviate this by for example, adding more training samples of that kind.

However, the decision map does not (yet) show the distance $d_{nD}(\mathbf{x})$ from a sample $\mathbf{x} \in D$ to its closest decision boundary $\partial DZ \subset D$ in the nD space. Rather, the map shows how close the *projection* $P(\mathbf{x})$ of $\mathbf{x}$ is to the *projection* $P(\partial DZ)$ of the decision boundaries. Simply put, for every pixel $\mathbf{y}$ having some color (label), the user can visually find the closest differently-colored pixel $\mathbf{y}'$. The distance

$$d_{2D}(\mathbf{y}) = \min_{\mathbf{y}' | f(\mathbf{y}) \neq f(\mathbf{y}')} \|\mathbf{y} - \mathbf{y}'\| \tag{8.2}$$

can thus be seen as a *projection* of the actual nD distance $d_{nD}(\mathbf{x})$ we are interested in. The two distances are not the same, given the local compression and stretching caused when

mapping the nD space to 2D by nonlinear projections such as t-SNE or UMAP [9, 148]. Note that Equation (8.2) is nothing but the so-called distance transform [62] of the set of pixels that constitute the decision zone boundaries ∂DZ.

Note that an *exact* computation of d_{nD} is impossible in general, since we do not have an analytic description of ∂DZ for typical classifiers. Hence, we next propose two classifier-independent heuristics to estimate d_{nD} (Sections 8.6.1 and 8.6.2) as well as a third, more exact, method and better suited for neural network classifiers, based on adversarial examples (Section 8.6.3). Figure 8.8 compares the 2D distance-to-boundary d_{2D} (computed by Equation (8.2), implemented using the fast distance transform method [30]), with two versions of the d_{nD} estimation we propose next, called d_{nD}^{img} and d_{nD}^{nn} respectively. In this figure, distances are encoded by a luminance colormap for illustration purposes. The decision zones and distance maps in Figure 8.8 depict a synthetic "Blobs" dataset with 60K observations sampled from a Gaussian distribution with 5 different centers (clusters), each one representing samples of one class and 50 dimensions. For classification, a simple LR model was used, so as to create simple-to-interpret decision boundaries, which are best as we next want to study the distance-to-boundary behavior. The same dataset was used to test the quality of the NNInv inverse projection in Chapter 7.

In Figure 8.8, we see that, while d_{2D} and d_{nD} are both low close to the decision boundaries and high deep in the decision zones, they have quite different local trends. For instance, points which have the same colors in Figure 8.8b, that is, are at the same distance-to-boundary (d_{2D}) in 2D, can have quite different colors in Figures 8.8c,d, that is, have different distances d_{nD} to the true nD decision boundaries. Hence, we cannot use d_{2D} as a 'proxy' to assess d_{nD}. We need to compute and show, d_{nD} to the user so one can estimate how close (or far) from a decision boundary an actual observation is.

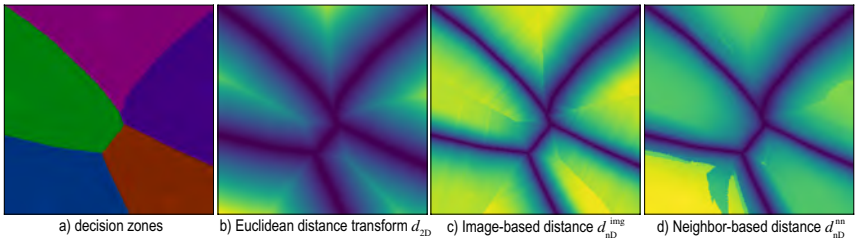


Figure 8.8: Dense map (**a**) and various distance-to-boundary maps (**b**–**d**) for Blobs dataset, computed using UMAP for P and NNInv for P^{-1}.

8.6.1 *Image-Based Distance Estimation*

For every pixel $\mathbf{q}$ in the dense map, we find the closest pixel $\mathbf{r}$ having a different label (Figure 8.9a). Let Q and M_s be the sets of nD samples in S_T that map to $\mathbf{q}$ and $\mathbf{r}$ respectively via P^{-1}. By construction, points in Q and M_s have thus different labels. Hence, the nD decision boundary ∂DZ lies somewhere between these point-sets. To estimate where, for every point pair ($\mathbf{x}_Q \in Q, \mathbf{x}_R \in R$), we compute the point $\mathbf{x}_{QR}$ along the line segment $(\mathbf{x}_Q, \mathbf{x}_R) \subset D$ where the classifier function f changes value, that is, turns from the label $f(\mathbf{x}_Q)$ to the label $\mathbf{x}_R$). For this, we use a bisection search, as we assume that f varies relatively smoothly between $\mathbf{x}_Q$ and $\mathbf{x}_R$. We use a maximum number of $T = 5$ bisection

steps, which proved to give good results in practice. We then estimate the distance of $\mathbf{q}$ to the closest decision boundary as the average

$$d_{nD}^{img}(\mathbf{q}) = \frac{1}{|Q||R|} \sum_{\mathbf{x}_Q \in Q, \mathbf{x}_R \in R} \|\mathbf{x}_Q - \mathbf{x}_{QR}\|. \tag{8.3}$$

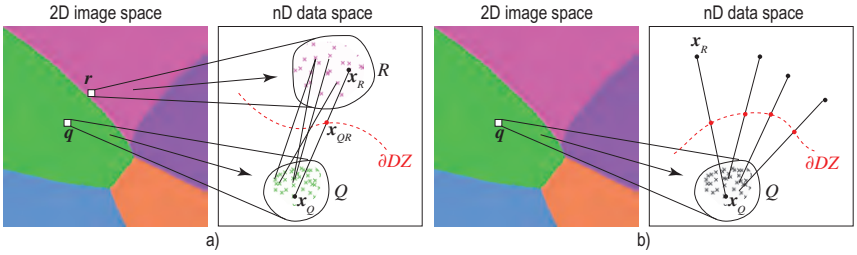


Figure 8.9: Estimation of distance-to-boundary d_{nD}^{img} (**a**) and d_{nD}^{nn} (**b**). See Sections 8.6.1 and 8.6.2.

Although Equation (8.3) is simple to evaluate, it can produce noisy estimations of d_{nD}. The main issue is that it assumes that the closest decision boundary to some point $\mathbf{q}$ in the 2D projection (i.e., pixel $\mathbf{r}$) corresponds, by the inverse mapping P^{-1}, to the closest decision boundary in nD to $P^{-1}(\mathbf{r})$.

8.6.2 *Nearest-Neighbor-Based Distance Estimation*

We can improve upon the dense map-based heuristic presented in Section 8.6.1 by disposing of the dense map as a tool to compute d_{nD}. Rather, we rely on searching the nD data directly for nearest-neighbor samples that have a different label, as follows (Figure 8.9b). For every pixel $\mathbf{q}$ in the dense map, let again Q be the set of nD samples that map to it via P^{-1}.

For each $\mathbf{x}_Q \in Q$, we next find the closest data point $\mathbf{x}_R \notin Q$ that is classified differently than $\mathbf{x}_Q$ and then again apply bisection to find where, along the line segment $(\mathbf{x}_Q, \mathbf{x}_R)$, the classifier f changes value. Finally, we compute $d_{nD}(\mathbf{q})$ by averaging all distances from $\mathbf{x}_Q$ to the corresponding bisection points x_{QR}. Formally put, we compute d_{nD} as

$$d_{nD}^{nn}(\mathbf{q}) = \frac{1}{|Q|} \sum_{\mathbf{x}_Q \in Q, \mathbf{x}_R = \underset{\mathbf{x} \notin Q | f(\mathbf{x}) \neq f(\mathbf{x}_Q)}{\arg\min} \|\mathbf{x} - \mathbf{x}_Q\|} \|\mathbf{x}_Q - \mathbf{x}_{QR}\|. \tag{8.4}$$

Estimating d_{nD} this way is more accurate than using Equation (8.3) since we do not rely on computing $\mathbf{x}_R$ using the possibly inaccurate dense map but directly use the nD points S. We implement Equation (8.3) by searching for nearest neighbors in nD space using the *kd*-tree spatial search structure provided by *scikit-learn* [1].

8.6.3 *Adversarial Based Distance Estimation*

The third proposed heuristic is based on adversarial examples [75, 197]. An adversarial perturbation ϵ of a data sample $\mathbf{x}$ can cause a trained classifier to assign a wrong label

to this so-called adversarial example $\mathbf{x} + \epsilon$, that is, a label different from the one that it assigns to the unperturbed sample $\mathbf{x}$. By definition, the minimal length $\|\epsilon\|$ of such a perturbation is the distance from $\mathbf{x}$ to the closest decision boundary to $\mathbf{x}$. Hence, we can compute the distance-to-boundary for a dense map pixel $\mathbf{q}$ by first gathering again all points Q that project to $\mathbf{q}$ and next averaging their distances to their closest nD boundaries computed as above. This defines

$$d_{nD}^{adv}(\mathbf{q}) = \frac{1}{|Q|} \sum_{\mathbf{x}_Q \in Q} \min_{f(\mathbf{x}_Q) \neq f(\mathbf{x}_Q + \epsilon)} \|\epsilon\|. \tag{8.5}$$

Compared to the distance-to-boundary heuristics given by Equations (8.3) and (8.4), Equation (8.5) yields a mathematically accurate distance to boundary, within the limits of sampling the perturbation space ϵ. In practice, this demands extensive computational resources, roughly three times more than evaluating Equation (8.4) and 30 times more than evaluating Equation (8.4). Moreover, the method is not guaranteed to yield a valid adversarial perturbation for all possible samples $\mathbf{x}$. Another limitation is that this approach is only suitable for classifiers f obtained through an iterative gradient-based optimization process, such as neural networks [75].

Figure 8.10a shows the dense maps (a) for the MNIST (top row) and FashionMNIST (bottom row) datasets respectively. Images (b-d) show the three distance-to-boundary functions d_{nD}^{img}, d_{nD}^{nn} and d_{nD}^{adv} given by Equations (8.3)–(8.5), respectively, visualized using the same luminance colormap as in Figure 8.8. Several observations follow.

First, we see that the nD distances d_{nD} roughly follow the patterns of the 2D Euclidean distances d_{2D}, that is, are low close to the 2D decision boundaries and high deeper inside the decision zones. However, the nD distances are far less smoothly varying as we get farther from the 2D boundaries. This indicates precisely the stretching and compression caused by P and P^{-1} mentioned earlier. Secondly, we see that d_{nD}^{img} is significantly less smooth than d_{nD}^{nn}. This is explained by the lower accuracy of the former's heuristic (Section 8.6.1). A separate problem appears for d_{nD}^{adv}: For the FashionMNIST dataset, the image shown is very dark, indicating very low d_{nD}^{adv} values for most pixels. Upon further investigation, we found that the neural network model trained for this case was too fragile—for almost every sample, an adversarial sample could be easily obtained. Moreover, as already mentioned, the cost of computing d_{nD}^{nn} is far larger than for the other two distance models. Given all above, we conclude that d_{nD}^{nn} offers the best balance of quality and speed and we choose next to use this distance-to-boundary model.

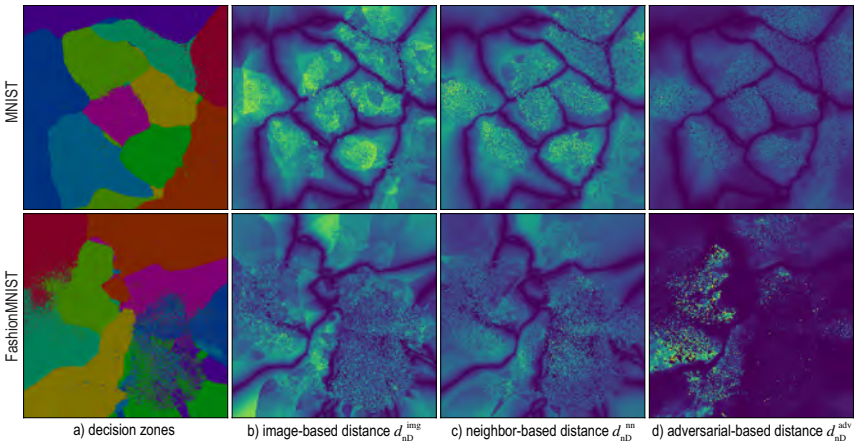


Figure 8.10: Dense map and distance maps for MNIST (top row) and FashionMNIST dataset (bottom row), with projection P set to UMAP and P^{-1} to NNInv respectively.

8.6.4 *Visualizing Boundary Proximities*

Visualizing the raw distance d_{nD} by direct luminance coding (Figure 8.10) does not optimally help us in exploring the regions of space that are *close* to decision boundaries. However, these are the areas one is most interested in, since these are the regions where classifiers may work incorrectly, by definition. For this, we apply a nonlinear transformation to d_{nD} to compress the high-value ranges and allocate more bandwidth to the low-value range. Also, we combine both decision zone information (shown by categorical colors in earlier figures) with the distance-to-boundary information in a single image. For this, we set the S (saturation) and V (value) color components of every pixel $\mathbf{q}$ in this image to

$$V(\mathbf{q}) = 0.1 + 0.9(1 - d_{nD}^{nn}(\mathbf{q})/d_{max})^{k_1} \tag{8.6}$$

$$S(\mathbf{q}) = S_{base}(1.0 - d_{nD}^{nn}(\mathbf{q})/d_{max})^{k_2} \tag{8.7}$$

Here, d_{max} is a normalization factor equal to the maximal value of d_{nD}^{nn} over the entire dense map; k_1 and k_2 are constants that control the nonlinear distance normalization; and S_{base} is the original saturation value of the categorical color used for $\mathbf{q}$'s label. The H (hue) component stays equal to the categorical-color encoding of the decision zone labels. Figure 8.11 shows the effect of k_1 and k_2 for the MNIST and FashionMNIST datasets. Compared to showing only the decision-zone information (Figure 8.11a), adding the distance information highlights (brightens) areas that are close in nD to the decision boundaries. Higher k_1 values highlight these zones more and darken areas deep in the decision zones more. Higher k_2 values strengthen this effect, as pixels close to decision boundaries become desaturated. This allows us to ensure that such pixels will be bright in the final images, no matter how dark the original categorical colors used to encode labels are.

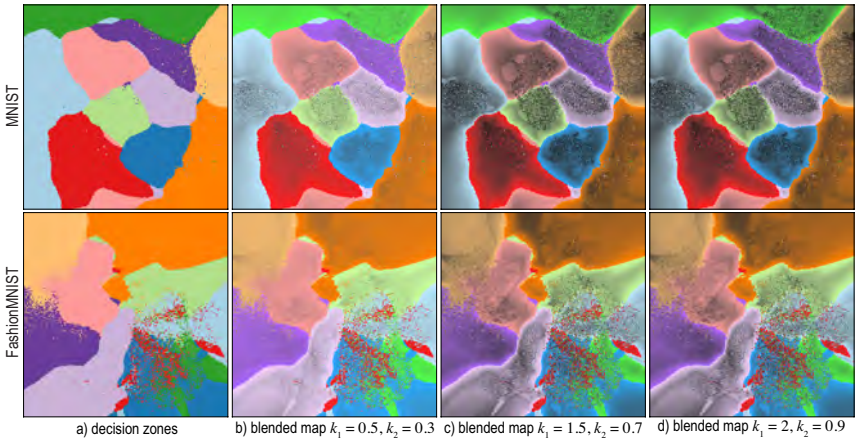


Figure 8.11: (**a**) Dense map for MNIST (top row) and FashionMNIST (bottom row) datasets. (**b**–**d**) Combined dense map and distance-to-boundary maps for different k_1 and k_2 values.

Figure 8.11 is to be interpreted as follows: Dark areas indicate data samples deep inside decision zones, that is, areas where a classifier will very likely not encounter inference problems. Bright areas indicate zones close to decision boundaries, where such problems typically appear and in which one should look for misclassifications and/or add extra labeled samples to improve training. Thin bright areas tell that the *n*D distance varies there much more rapidly than the perceived 2D (image-space) distance, so the projection *compresses* distances there. These are areas on which one will typically want to zoom in, to see more details. In contrast, thick bright areas tell that the *n*D distance varies there slower than the perceived 2D distance, so the projection *stretches* distances there. Such areas normally do not require zooming to see additional details.

Figure 8.12 shows a different use-case for distance maps. Atop of the distance maps shown in Figure 8.11 ($k_1 = 2, k_2 = 0.9$), we now plot the misclassified points for MNIST and FashionMNIST, encoding their respective distance-to-boundary d_{nD} in opacity. Misclassifications which are close to decision boundaries show up thus as opaque white, while those deeper in the decision zones show up half-transparent. We see now that most misclassifications occur either close to the smooth decision boundaries (MNIST) or atop of small decision-zone islands (FashionMNIST). Since islands, by definition, create decision boundaries, it follows that, in both cases, misclassifications predominantly occur close to decision boundaries. Hence, decision boundaries can serve as an indicator of areas prone to misclassifications, thus potential targets for refining the design of a classifier for example, by data annotation or augmentation.

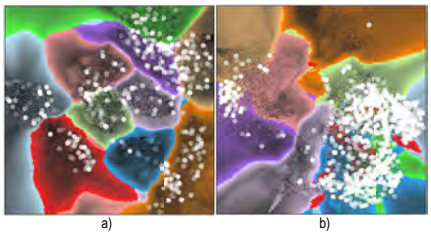


Figure 8.12: Misclassifications with opacity coding distance-to-boundary for (**a**) MNIST and (**b**) FashionMNIST datasets.

Enridged Distance Maps

Figure 8.11 encodes distance-to-boundary by luminance and saturation, which are good visual variables for *ordinal* tasks, for example, estimating which points are closer or farther from decision boundaries. However, this encoding is less suitable for *quantitative* tasks, for example, estimating equal-distance points or how much farther (or closer) a given point is to its closest decision boundary than another point. We address these tasks by using enridged cushion maps [221]. For this, we first slightly smooth d_{nD} by applying a Gaussian filter with radius K pixels. Next, we pass the filtered distance through a periodic transfer function $f(x) = (x \bmod h)/h$ and use the resulting value $f(d_{nD})$ instead of d_{nD} to compute S and V via Equations (8.6) and (8.7). Note that the transfer function f is only *piece wise* continuous [221], and requires *smooth* signals as input to yield visually smooth cushions. Since our high-dimensional distance d_{nD} is not overall smooth, due to the already discussed inherent projection errors and also due to the numerical approximations used when computing it (see Sections 8.6.1–8.6.3), filtering is required. Besides filtering, a second difference between our approach and the original technique [221] is that we visualize directly the *distance*, whereas they visualized a shaded *height plot* of the distance. We choose in our case to visualize the distance directly as this is faster to compute and more robust to noise—height plot shading requires normal computations which, given our inherently noisy distance estimations, can easily become unreliable.

Figure 8.13 shows the results for the MNIST and FashionMNIST datasets. Each apparent luminance band in the image shows points located within the same distance-to-boundary interval. Dark thin bands are analogous to contours or isolines, of the distance-to-boundary. Finally, the thickness of the bands indicate distance compression (thin bands) respectively distance stretching by the projection (thick bands). We also see how increasing the filter radius K progressively smooths the image, removing projection artifacts and making it easier to interpret.

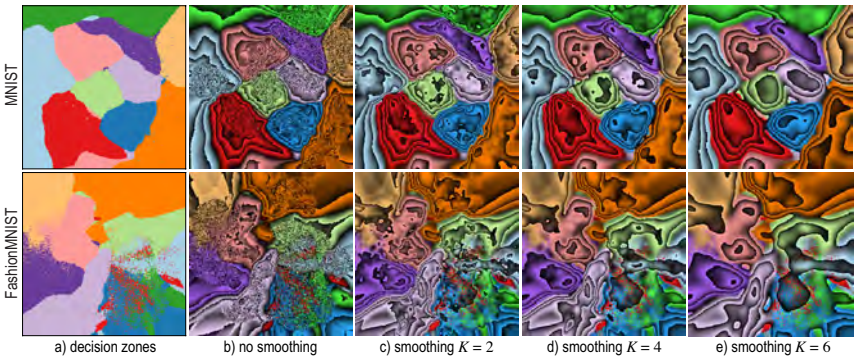


Figure 8.13: Enridged distance maps for MNIST (top row) and FashionMNIST (bottom row) datasets. Images (**b**–**e**) show the progressive noise-smoothing effect of the filter radius K.

8.7 DISCUSSION

We summarize our findings and insights concerning the construction and interpretation of classifier decision maps as follows.

Novelty: While dense maps have been used earlier in high-dimensional data visualization to analyze projection quality [9, 133], they have not been used for explicitly visualizing the decision zones of any classifier. Besides showing the actual decision zones by color coding, we also compute and show the actual distance-to-boundary, which highlights zones close to boundaries, where a classifier is most prone to misclassify data. The work of Schultz *et al.* [180] is closest to our work and, to our knowledge, the only other method (apart from ours) which aims to explicitly visualize classifier decision zones. However, several important differences between our work and that exist, as follows:

- Computation of inverse projection P^{-1}: In Schultz *et al.* [180], this is done by extending non-parametric projections P to parametric forms, by essentially modeling P as the effect of several fixed-bandwidth Gaussian interpolation kernels. This is very similar to the way iLAMP works. However, as shown in Chapter 7, iLAMP is far less accurate and far slower than other inverse projection approaches such as NNinv. In our work, we let one freely choose how P^{-1} is implemented, regardless of P. In particular, we use the deep-learning inverse projection NNinv which is faster and more accurate than iLAMP;

- Supervised projections P: In Schultz *et al.* [180], the projection P is implemented using so-called discriminative dimensionality reduction which selects a subset of the nD samples to project, rather than the entire set, so as to reduce the complexity of DR and thus make its inversion more well posed. More precisely, label information for the nD samples is used to guide the projection construction. While this, indeed, makes P easier to invert, we argue that it does not parallel the way typical practitioners work with DR in machine learning. Indeed, in most cases, one has an nD dataset and projects it fully, to reason next about how a classifier trained on that dataset will behave. Driving P by class label is, of course, possible but risky, since P next does not visualize the *actual* data space. Moreover, discriminative DR

is quite expensive to implement ($O(N^2)$ for N sample points). Note that our outlier filtering (Section 8.5) achieves roughly the same effect as discriminative DR but at a lower computational cost and with a very simple implementation;

- Distance to boundary: In Schultz *et al.* [180], this quantity, which is next essential for creating dense decision boundary maps, is assumed to be given by the projection algorithm P. Quoting from Schultz *et al.* [180]: "We assume that the label $f(\mathbf{x})$ is accompanied by a nonnegative real value $r(\mathbf{x}) \in \mathbb{R}$ which scales with the distance from the closest class boundary." Obviously, not all classifiers readily provide this distance. Moreover, getting hold of this information (for classifiers which provide it) implies digging into the classifier's internals and implementation. We avoid such complications by providing ways to estimate the distance to boundary generically, that is, considering the classifier as a black box (Section 8.6).
- Computational scalability: Schultz *et al.* [180] does not discuss the scalability of their proposal, only hinting that the complexity is squared in the number of input samples. Complexity in the resolution of the decision maps is not discussed. In contrast, we detail our complexity (see *Scalability* below).

Genericity: We can generically construct decision maps, including the estimation of distance-to-boundary, for datasets in any dimension and for any classifier. This makes our techniques easily usable for a wide range of applications in machine learning.

Best techniques: We evaluate the construction of dense maps using 28 direct projection techniques and 3 inverse projection techniques respectively. To limit the amount of work required to analyze hundreds of classifier-projection combinations, we designed a two-phase experiment where we pre-select the best projections (using a simple classification problem) to study next in detail. t-SNE and UMAP appear to be the best projections for constructing dense maps in terms of recognizability of decision boundaries in the produced patterns, limited errors (spurious islands) and concentration of confusion zones (misclassifications). Since UMAP has similar properties with t-SNE but is significantly faster, we label it as the optimal candidate for this task. For the inverse projection, NNInv is the technique of choice.

Replicability and extensibility: To be useful, our work on evaluating projection-based dense maps must be accessible, replicable and extensible. All involved materials and methods (projections, datasets, dense maps, classifiers, automated workflow scripts) are available online (`mespadoto.github.io/dbm/`). We intend to organically extend this repository with new instances along all above-mentioned dimensions.

Scalability: Computational scalability of our method is influenced by the complexity of the direct projection technique P and inverse projection technique P^{-1}; and the number N of high-dimensional points and resolution (number of pixels M_s) in the decision boundary map image. Summarizing our method, we (1) use P to project N samples to 2D; (2) for each of the M_s pixels, we use P^{-1} to infer its high-dimensional sample; and (3) use the classifier f to infer the label of that sample, which we finally draw at the respective pixel. Denote the cost of projecting a single sample by C_P; the cost of inverting the projection at a single pixel by $C_{P^{-1}}$; and the cost of classifying a single sample by C_f, respectively. Then, the cost of our entire method is $NC_P + R(C_{P^{-1}} + C_f)$. Several insights can be drawn from this. First, our method is linear in the resolution of the decision boundary image. This is the dominant cost factor, since the number of pixels M_s in a typical decision map is larger than the number of samples N (see for

instance the example datasets discussed in Section 8.3. If using a fast inverse projection P^{-1} such as NNinv and deep learning classifiers f, which are both practically linear in N, the entire pipeline can be run to construct decision maps in a few seconds on a typical desktop PC.

Limitations: Constructing accurate decision maps is an even harder problem than the already difficult task of accurately projecting high-dimensional data into 2D. While our study showed that the (UMAP, NNInv) combination of direct and inverse projection techniques yields good results in terms of visually easy-to-identify decision zones, we cannot guarantee such results for *any* high-dimensional dataset and classifier combination. More precisely, errors caused by the direct and/or inverse projections can still manifest themselves as jaggy boundaries and/or islands present in the resulting decision maps. These errors can be decreased by further filtering wrongly projected points that lead to poor neighborhood preservation (Section 8.5). Also, showing the distance-to-boundary (Section 8.6) can highlight the presence of remaining errors.

Applications: Currently, our decision maps can only show how a classifier partitions the high-dimensional space into decision zones corresponding to its different classes. This can help the practitioner to better *understand* the behavior of such a classifier but not directly to *improve* the classifier. Recent separate work has shown that projections are effective tools for data annotation purposes, that is, creating new labeled samples for increasing the size of training sets with little human effort by visually extrapolating labels of existing samples to close unlabeled ones [19]. Our decision maps can very likely help such data annotation by informing the user how to perform this visual extrapolation so as not to cross decision boundaries. Separately, our decision maps could help in data augmentation tasks by showing which areas in the high-dimensional space are densely populated by misclassifications (Figure 8.12). Selecting 2D points in such areas can be used to easily generate high-dimensional samples, via the inverse projection, to augment a training set. We consider all these directions for future work.

8.8 CONCLUSIONS

We have presented an end-to-end pipeline for constructing decision maps for understanding the behavior of classifiers in machine learning. To this end, we have evaluated 28 well-known projections on a two-class, respectively ten-class, subset of a well-known machine learning benchmark, using four well-known classifiers. Our evaluation shows wide and to our knowledge, not yet known, differences between the behavior of the studied projections. Using a visual analytics methodology, we next refined our analysis to a small set of five high-quality projections and found that t-SNE and UMAP perform best for this task. We next proposed a filtering approach to decrease the adverse impact of inherent projection errors and thereby construct more faithful, less noisy, decision maps. Finally, we proposed to visualize the distance-to-boundary of every decision map point (computed by three different approximations we propose) and thereby augment the amount of information on classifier behavior that these maps convey.

Several future work directions are possible. First and foremost, we plan to use our decision maps to support data annotation and/or augmentation tasks to help practitioners in designing better classifiers with limited effort. Secondly, decision maps can be used to compare the behavior of different classifiers addressing the same problem. Finally, our technique can be adapted to visualize high-dimensional spatial partitioning schemes,

such as n-dimensional Voronoi diagrams and thereby give more insights in the behavior of high dimensional data spaces.

In Chapter 9, we present a technique for drawing multivariate functions, in the context of optimization problems, which uses direct and inverse projections. This new technique can be seen as the "continuous" version of the decision boundary map technique presented in this chapter.

9 VISUALIZING OPTIMIZATION PROBLEMS

9.1 INTRODUCTION

Operations Research (OR), also called Management Science, plays a crucial role in many industries, from logistics to finance. Although its origins as a discipline date from the 1950s, with the development of the Simplex algorithm for Linear Programming [42, 98], it is a field in constant development since them. The OR practitioner has many tools at their disposal which improve their productivity, such as algebraic modeling languages like GAMS [27], AMPL [68] and more recently, JuMP [48], which enable the use of notation very close to the mathematical definition of optimization problems. Yet, to our knowledge, there are no well-established *visualization* tools that help understand multivariate objective functions with respect to the decision variables and constraints (if any) of the problem. Such tools are important as a complement to more formal tools for getting an overall understanding of how an objective function behaves subject to its many parameters.

In this chapter we describe a technique called OptMap, which is an image-based visualization tool that uses multidimensional projections to enable the OR practitioner to literally see the decision variables and constraint spaces using a two-dimensional dense map, regardless of the number of variables and constraints in the problem. We show that OptMap can be used in several ways, such as a debugging aid to help diagnose errors in the definition of constraints; as a tool to provide insight of the optimizer's inner workings, by plotting the path taken from a starting point to a solution; and as a general tool to visually explore the high-dimensional space of the decision variables in terms of objective function value and constraint feasibility.

OptMap aims to cover the following aspects, which, to our knowledge, are not achieved by existing visualization techniques in the context of optimization:

Quality (C1): We provide high-quality visualizations, that encode information at every available screen pixel, by using a combination of dense maps, direct, and inverse projection techniques;

Genericity (C2): We can handle many kinds of optimization problems for single-valued objective functions. The only requirements we impose are that the user provides implementations of the objective function, constraints (if any), and the domain of each variable;

Simplicity (C3): Our technique is based on existing projection techniques which have a straightforward implementation, allowing easy replication and deployment;

Ease of use (C4): Our technique has few hyperparameters, all with given presets. In most cases, users do not have to adjust those to obtain good results;

This chapter is based on publication [54]

Scalability (C5): By using a fast projection technique and caching results when possible, our method is fast enough to allow its use during the rapid development-test cycle of optimization models.

We structure the chapter as follows. Section 9.2 presents the notations used and discusses related work on visualization for multivariate functions and optimization problems, Section 9.3 details our method. Section 9.4 presents the results that support our contributions outlined above. Section 9.5 discusses our proposal. Section 9.6 concludes the chapter.

9.2 BACKGROUND

Related work concerns optimization techniques (Section 9.2.1) and visualization (Section 9.2.2). Please refer to Chapter 3 for a thorough discussion of dimensionality reduction techniques.

9.2.1 *Optimization*

Optimization problems come in many forms with respect to the kind of function to be optimized, the type of decision variables, and the existence of constraints. Functions are typically grouped into linear, convex and non-convex. *Linear* functions are of the form $f(\mathbf{x}) = \mathbf{a}\mathbf{x} + b$, which defines a hyperplane; *convex* functions can have many forms, but can be defined as those where the set of points above their graph forms a convex set; *non-convex* functions are neither linear nor convex (linear functions are also convex). Decision variables can be continuous or discrete: Problems with only discrete variables are called Integer Programs (IP) [77], whereas problems with a combination of discrete and continuous variables are called Mixed Integer Programs (MIP). Lastly, problems can be constrained or unconstrained. Constraints can be characterized just as functions (linear, convex, non-convex). Additionally, we have box constraints, which are simple restrictions on the variables' domains. Problems with continuous variables, linear objective functions, and linear constraints are called Linear Programs (LP). Other problems are solved by Nonlinear Programming (NLP) techniques.

We next define a few notations for optimization problems. Let $f : \mathbb{R}^n \rightarrow \mathbb{R}$ be some function to be minimized. Let $\mathbf{x} = (x^1, \ldots, x^n)$, $x^i, 1 \leq i \leq n$, be an n-dimensional vector of n decision variables x^i. Decision variables can be any combination of discrete ($\mathbb{Z}$) and continuous ($\mathbb{R}$). Let O be the optimization problem described as

$$\begin{aligned} &\text{minimize} \quad f(\mathbf{x}) \\ &\text{subject to} \quad \mathbf{x} \in S \end{aligned} \tag{9.1}$$

where S is the *feasible set* of all points that can be considered as valid for the optimization problem. For unconstrained problems, S is $\mathbb{R}^n$. For constrained problems there is a set of K constraint functions $c_k(\mathbf{x}) \in \{0, 1\}, k \in 1, ..., K$, where 0 means that the point $\mathbf{x}$ is infeasible with respect to the constraint c_k. That is, for constrained problems, the feasible set is defined as $S = \{\mathbf{x} : \prod c_k(\mathbf{x}) = 1, k \in 1, ..., K\}$.

Solvers are algorithms that find one of several (approximate) solutions to a problem O. To do this efficiently, solvers use the characteristics of the problem, such as the type of decision variables, objective function and constraints, and employ adequate heuristics to avoid exploring all possible $\mathbf{x} \in S$, which would be impractical in most cases. Probably the most popular solver algorithm is the Simplex [42, 98], used for linear problems

and implemented by software such as Clp [66], Cbc [67] and GLPK [130]. For non-linear optimization problems there are other algorithms such as Gradient Descent, Nelder-Mead [145] and L-BFGS [118], to name a few. Many solvers work iteratively, *i.e.*, start from a given point $\mathbf{x}_0$ and evolve this point until sufficiently close to the solution of O.

A *solution* is an n-dimensional point found by the solver which meets the criteria of being feasible ($\mathbf{x} \in S$) and *optimal*. The definition of optimality depends on the type of problem and solver used: For linear functions with linear constraints, solvers are guaranteed to find a *global optimum* solution, which means that no other n-dimensional point provides a lower value for the objective function f, given those constraints. For non-linear functions, solvers may return different *local optima*, depending on the starting point $\mathbf{x}_0$ used and the shape of the objective function.

Lastly, a solver may provide the user with a *trace*, or *path to solution*, which is the set of all n-dimensional points where it evaluated the objective function, starting from $\mathbf{x}_0$ and ending with the solution, if one was found, else ending with the last point evaluated by the solver.

9.2.2 *Visualization*

The visualization of 2D functions $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ is usually done by means of 3D height plots, contour plots, or color (heatmap) plots. For functions $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with more than two variables ($n > 2$), there are far fewer options, with Hyperslice [220] being a notable one. Hyperslice presents a multidimensional function v as a matrix of orthogonal two-dimensional slices, each showing the restriction of f to one of the 2D subspaces in R^n, using the 2D function plotting outlined earlier (contour plots, color plots, 3D height plots).

Visualizing constrained optimization problems is similar to the above, since not only the function has to be visualized but constraint feasibility as well. Most techniques used for this are based on overlaying contour plots with constraint information, with one case where image-based techniques are used [219]. Still, such techniques cannot work with more than two dimensions ($n > 2$).

The authors of iLAMP [5] used direct and inverse projection techniques applied to non-linear optimization problems, to help users interactively identify good starting points for optimization problems. However, iLAMP is computationally expensive, and has quite a number of free parameters the user needs to set. The NNInv method described in Chapter 7 accelerates inverse projections by over two orders of magnitude as compared to iLAMP by deep learning the inverse projection function P^{-1}. The same deep learning idea was also used to accelerate the direct projection P by Neural Network Projection (NNP), described in Chapter 5. NNInv was used in the image-based (dense map) technique to visualize the decision boundaries for Machine Learning classifiers, described in Chapter 8, for problems with arbitrary dimension. That method can be conceptually seen as the visualization of a function $f : \mathbb{R}^n \rightarrow C$, where f is a classifier for nD data and M_c is a class (label) set. Here, we share the same idea of using a dense pixel map to visualize high-dimensional functions. However, we treat *real-valued* functions f rather than classifiers; and our aim is understanding optimization problems rather than understanding the output of a classifier, so we treat a different problem and use-case.

9.3 METHOD

We next describe the OptMap technique. Figure 9.1 shows a high-level diagram of OptMap, with each step described in detail next.

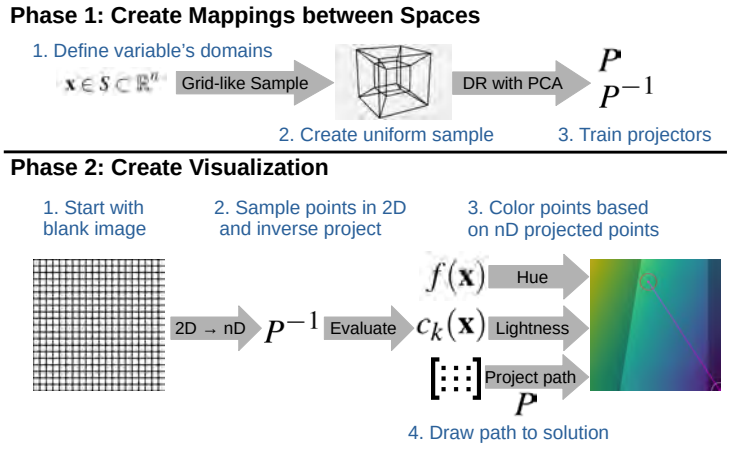


Figure 9.1: OptMap Pipeline.

1. Define variables domains: the user specifies the domain of each variable x_i for $f(x_1, \ldots, x_n)$. When the range is the entire real axis $\mathbb{R}$, we suggest selecting a reasonable finite range to avoid having a too coarse sampling for that variable;

2. Sample data: We uniformly sample the ranges defined above for each variable, yielding a regular sample grid $G^n \subset \mathbb{R}^n$. We constrain the maximum number of sample points N_{max} to avoid combinatorial explosion. In this chapter, we used $N_{max} = 5\text{M}$ for all experiments. We evaluate f on G^n and call the resulting dataset D.

3. Create mappings: we use PCA [97] trained on D to create the mappings P and P^{-1} from $\mathbb{R}^n$ to $\mathbb{R}^q$, respectively from $\mathbb{R}^q$ to $\mathbb{R}^n$.

4. Create a 2D grid: We create an uniform grid $G^2 \subset \mathbb{R}^2$ similar to G^n. Simply put, G^2 is a pixel image of some fixed resolution (set to 800^2 for the experiments in this chapter). Next, we use the trained P^{-1} to map each grid point (pixel) $\mathbf{p} \in G^2$ to a high-dimensional point $\mathbf{x} \in \mathbb{R}^n$. Finally, we evaluate the objective function $f(\mathbf{x})$ and optional constraints.

5. Color pixels: We color all pixels $\mathbf{p} \in G^2$ by the values of $f(P^{-1}())$, using a continuous color map, set to the Viridis color map [92] in the examples in this chapter. Additionally, we set the luminance of $\mathbf{p}$ to reflect $f(P^{-1}(\mathbf{p}))$'s membership of the constraint-set S, thereby indicating constraint feasibility.

6. Draw path to solution: if the solver provides the trace to a solution, as defined in Section 9.2.1, we can draw it in the 2D grid by projecting them using P.

9.4 RESULTS

We next show several experiments that show how our OptMap technique performs in different scenarios. First, we use OptMap to visualize high-dimensional functions that have a *known* shape (Section 9.4.1). Since we know the ground truth, we can check how OptMap performs. Next, test our method on several unconstrained and constrained optimization problems (Secs. 9.4.2 and 9.4.3 respectively) and show the added value OptMap provides in these actual use cases.

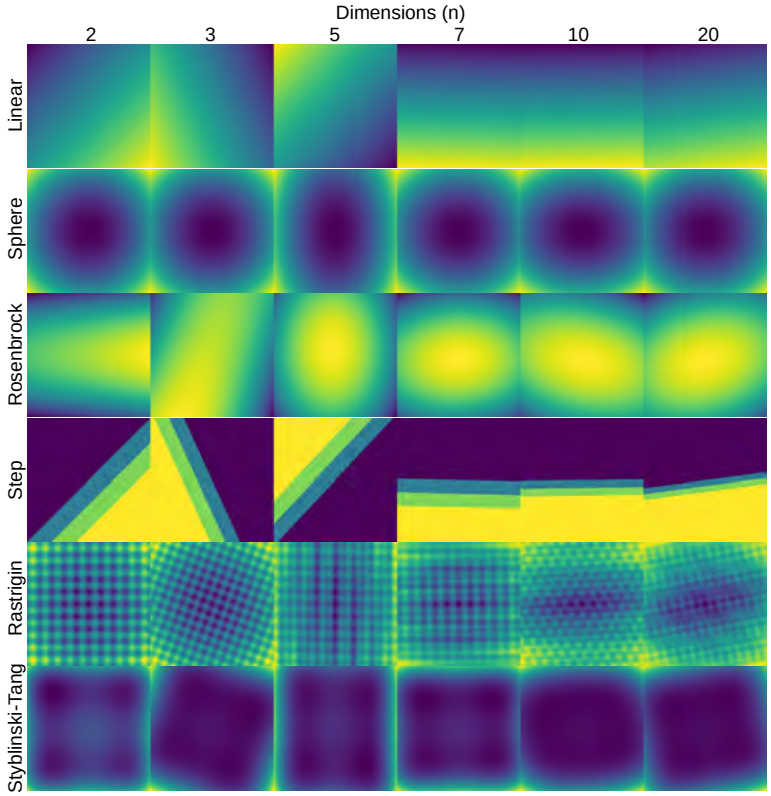


Figure 9.2: Dense maps for functions with known shape as defined in Table 19, with increasing dimensionality $n > 2$. Compare these with the ground-truth maps for $n = 2$.

9.4.1 *Ground-Truth Functions*

We use the six functions f listed in Table 19 to test OptMap. The corresponding dense maps, computed as explained in Section 9.3, are shown in Fig. 9.2. In all cases, the domain used for all variables was $x_i \in [-5, 5]$. All these functions have a predictable shape and also generalize to many dimensions. We created dense maps using increasing numbers of dimensions $n \in \{2, 3, 5, 7, 10, 20\}$. The dense map for $n = 2$ was created for reference only, without using OptMap. Indeed, for $n = 2$, we can directly visualize f, *e.g.*, by color coding, similar to [220]. Showing these maps for $n = 2$ is however very useful. Indeed,

Table 19: Definition of n-dimensional selected functions for ground-truth testing.

Function Name	Definition	Function Name	Definition
Linear	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i$	Rosenbrock [173]	$f(\mathbf{x}) = \sum_{i=1}^{n-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right]$
Sphere	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	Rastrigin [165]	$f(\mathbf{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i) \right]$, where: $A = 10$
Step	$f(\mathbf{x}) = \begin{cases} 0 & \sum_{i=1}^{n} x_i < 0 \\ 2 & \sum_{i=1}^{n} x_i < 2 \\ 4 & \sum_{i=1}^{n} x_i < 4 \\ 5 & otherwise \end{cases}$	Styblinski-Tang [194]	$f(\mathbf{x}) = \frac{\sum_{i=1}^{n} x_i^4 - 16x_i^2 + 5x_i}{2}$

Table 20: Solvers used for unconstrained problems.

Solver Type	Solver
Gradient-free	Nelder-Mead [145]
	Simulated Annealing
Gradient required	Gradient Descent
	Conjugated Gradient [79]
	L-BFGS [118]
Hessian required	Newton

(1) for $n = 2$, we can show f *directly*, without any approximation implied by OptMap; and (2) given the functions' expressions (Table 19), we know that they behave similarly regardless of n. Hence, if for $n > 2$ OptMap produces images similar to the ground truth ones for $n = 2$, we know that OptMap works well. And indeed, Fig. 9.2 shows us exactly this – the OptMap images for $n > 2$ are very similar to the ground-truth ones for $n = 2$. The differences imply some distortion and rotations, which, we argue, are are expected and reasonably small, given the inherent information loss when mapping a nD phenomenon to 2D.

9.4.2 *Unconstrained Problems*

We next use OptMap to show how different solvers perform when solving different unconstrained problems (that is, variants of Equation (9.1)). For this, we select a subset of the functions defined in 19, namely Sphere, Rastrigin and Styblinski-Tang functions, with varying dimensionality n. We use the solvers listed in Table 20, grouped by solver type, namely whether it is gradient-free or if it requires a gradient or a Hessian. In Figure 9.3 we use OptMap to show the *trace* provided by each solver, *i.e.*, all the points evaluated by the solver to get to the solution. We see that for a simple function with a global optimum (Sphere) most solvers find an optimal solution, except for the gradient-free methods, which seem to struggle with the high-dimensionality of the problem ($n = 20$ dimensions). For the Styblinski-Tang function, we see different but close optima were found by most solvers. We also see that both gradient-free methods evaluated many more points than the other methods, but that Nelder-Mead kept moving in the right

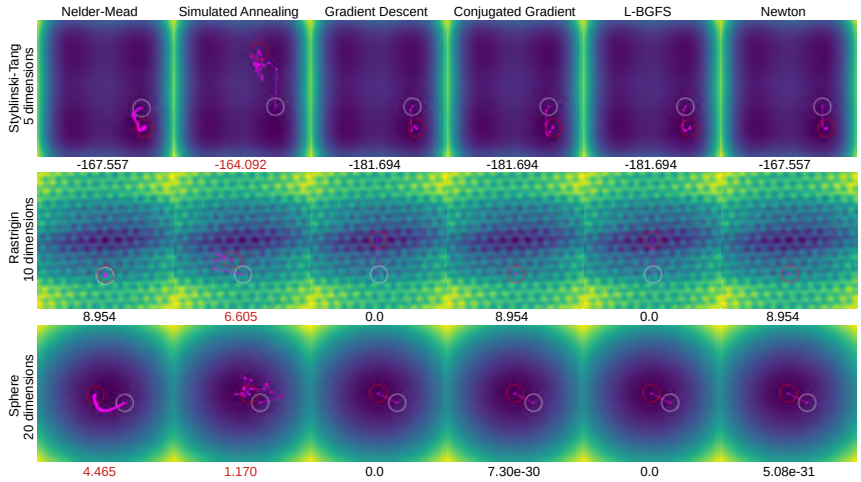


Figure 9.3: Dense maps created with OptMap for unconstrained problems using the solvers defined in Table 20 and some of the functions defined in Table 19. White circles indicates starting points (random vectors in 5, 10 and 20 dimensions respectively). Red circles indicate optimal points found by the solver. The magenta lines and points show each point evaluated by the solver to get to the solution. The numbers below each image indicate the value of the objective function at the solution; red values indicate that the solver failed to find an optimal solution.

direction. For the same problem, Simulated Annealing had problems converging to an optimal solution and eventually gave up. For the Rastrigin function, which has many optima, we see that only Gradient Descent and L-BFGS managed to find the solution in a straightforward way, while the other methods converged to the wrong solution or did not converge.

9.4.3 *Constrained Problems*

We next show how our OptMap performs when dealing with constrained optimization problems – that is, finding the minimum of some n-dimensional function f whose variables who are constrained as described in Section 9.2.1. Table 21 shows the definition of constrained problems (objective functions and constraints) we used. The first three problems used are very common in the optimization literature [77]. The last two problems use the same Sphere and Styblinski-Tang functions defined earlier, but with nonlinear constraints added to them. Figure 9.4 shows how OptMap visualizes the problem space and solution for each problem. Unfortunately, the solvers used in this experiment, namely Clp [66], Cbc [67], GLPK [130] and Ipopt [211] do not provide trace information to be drawn through the algebraic modeling language we used, JuMP [48], so we only draw the straight-line path from the (randomly chosen) starting point to solution.

For all problems, we can see the relationship between the objective function and the constraints of the problem, which provides insight on how close to boundary conditions the solutions are. For example, in in the problems Schedule, Sphere and Styblinski-Tang, we see that the solution found is at the boundary of one or more constraints. This is not the case for the Diet and Knapsack problem, which indicates that some tuning to the

solver's settings may be required to obtain better results, or even some adjustments to the problem definition may be done, such as the relaxation of some constraints.

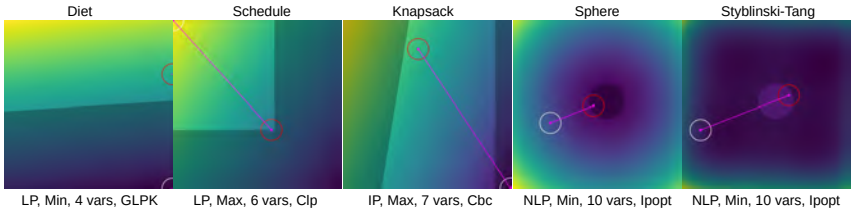


Figure 9.4: Dense maps created with OptMap for the constrained problems defined in Table 21. White circle indicates starting points (zero vector). Red circle indicate optimal points found by the solver. Magenta lines shows the path from the starting point to the solution, and darker areas indicate unfeasible regions. The text below each image indicate type of problem, direction (minimization or maximization), number of variables, and solver used.

9.4.4 *Performance*

OptMap's computation time can be divided in two phases (Fig. 9.1): In phase 1, most of the time is spent while running PCA for the sampled points in the grid G^n to define the mapping between the nD and 2D spaces. This is a task that has to be done only once for a given function f and can be reused afterwards when one changes the solver. In phase 2, most of the time is spent by evaluating the objective function f and its constraints. Since the evaluation of functions is usually very fast and the pixel grid G^2 is of limited size (800^2 in our experiments), phase 2 takes only a few seconds to run on our platform. Table 22 shows the time it takes to run PCA in phase 1 for $N_{max} = 5$M points, where we see that time increases very quickly with dimensionality. However, since phase 1 is required to be run only once, and since it takes at most a few minutes to run even with a high number of dimensions n (see Table 22), we argue that this is not a crucial limitation of OptMap.

9.5 DISCUSSION

We discuss next how OptMap performs with respect to the criteria laid out in Section 9.1.

Quality (C1): Figures 9.2, 9.3 and 9.4 show examples of the quality of the visualizations and the kind of insight they can provide for optimization problems. Our dense maps are pixel-accurate, in the sense that they show *actual* information inferred from the nD function f under investigation at each pixel, *without* interpolation. This is in contrast with many other dimensionality reduction methods which either show a *sparse* sampling of the nD space (by means of a 2D scatterplot), leaving the user to guess what happens between scatterplot points; or use interpolation in the 2D *image* space to 'fill' such gaps [47, 133, 188], which creates smooth images that may communicate wrong insights, since we do not know the underlying projection is continuous.

Genericity (C2): We show how our technique performs for optimization problems with varying nature, complexity, and dimensionality. We also show that our method can be used simply for visualizing high-dimensional, continuous, functions by a *single*

Table 21: Definition of constrained optimization problems.

Name	Definition
Diet	$\begin{aligned} \text{minimize} \quad & 0.14x_1 + 0.4x_2 + 0.3x_3 + 0.75x_4 \\ \text{subject to} \quad & 23x_1 + 171x_2 + 65x_3 + x_4 \geq 2000.0, \\ & 0.1x_1 + 0.2x_2 + 9.3x_4 \geq 30.0, \\ & 0.6x_1 + 3.7x_2 + 2.2x_3 + 7x_4 \geq 200.0, \\ & 6x_1 + 30x_2 + 13x_3 + 5x_4 \geq 250.0, \\ & x_1, x_2, x_3, x_4 \geq 0.0 \end{aligned}$
Schedule	$\begin{aligned} \text{maximize} \quad & 300x_1 + 260x_2 + 220x_3 + 180x_4 - 8y_1 - 6y_2 \\ \text{subject to} \quad & 11x_1 + 7x_2 + 6x_3 + 5x_4 \leq 700.0, \\ & 4x_1 + 6x_2 + 5x_3 + 4x_4 \leq 500.0, \\ & 8x_1 + 5x_2 + 5x_3 + 6x_4 - y_1 \leq 0.0, \\ & 7x_1 + 8x_2 + 7x_3 + 4x_4 - y_2 \leq 0.0, \\ & y_1 \leq 600.0, \\ & y_2 \leq 650.0, \\ & x_1, x_2, x_3, x_4, y_1, y_2 \geq 0.0 \end{aligned}$
Knapsack	$\begin{aligned} \text{maximize} \quad & 60x_1 + 70x_2 + 40x_3 + 70x_4 + 20x_5 + 90x_6 \\ \text{subject to} \quad & x_1 + x_2 - 4y \geq 0.0, \\ & x_5 + x_6 + 4y \geq 4.0, \\ & 30x_1 + 20x_2 + 30x_3 + 90x_4 + 30x_5 + 70x_6 \leq 2000.0, \\ & x_3 - 10x_4 \leq 0.0, \\ & x_1, x_2, x_3, x_4, x_5, x_6, y \geq 0.0, \\ & x_1, x_2, x_3, x_4, x_5, x_6 \leq 10.0, \\ & y \leq 1.0, \\ & x_1, x_2, x_3, x_4, x_5, x_6, y \in \mathbb{Z} \end{aligned}$
Sphere	$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{10} x_i^2 \\ \text{subject to} \quad & \sum_{i=1}^{10} x_i^2 \geq 5.0 \end{aligned}$
Styblinski-Tang	$\begin{aligned} \text{minimize} \quad & \frac{\sum_{i=1}^{10} x_i^4 - 16x_i^2 + 5x_i}{2} \\ \text{subject to} \quad & \sum_{i=1}^{10} x_i^2 \geq 5.0 \end{aligned}$

Table 22: Time to project N_{max} = 5M points with different dimensionalities n using PCA.

Dimensions n	Time (sec)
3	1.19
5	0.85
7	1.45
10	2.38
20	6.62
50	32.74
100	108.71

2D image, in contrast to multiple images that have to be navigated and correlated by interaction [220]. We also show that our technique is independent of the optimization solvers being used;

Simplicity (C3): We use PCA for direct and inverse projections, which is a very well known, simple, fast, and deterministic projection method. OptMap's complete implementation has about 250 lines of Julia code. Note that we also experimented with other methods for the direct projection – namely, t-SNE as learned by NNP – and inverse projection – namely, NNInv, obtaining good results. However, for the optimization problems presented in this chapter, PCA yielded better results (based on ground truth comparison). Since PCA is also simpler and faster than NNP and NNInv, we preferred it in our work.

Ease of use (C4): Apart from the timing experiment in Section 9.4.4, we executed all experiments using the same maximum number of sample points N_{max} with good results, which shows that the technique requires little to no tuning to work properly;

Scalability (C5): Section 9.4.4 shows that our method is highly scalable with the number of sample points and dimensions, which enables its interactive usage during the development cycle of optimization models.

Limitations: The projected points, such as the starting, trace, and solution points, are placed in the 2D image space at *approximate* positions, due to the inherent discrete nature of the pixel grid G^2. This can cause situations such as the one in Fig. 9.4 (Sphere problem), where the optimal point found by the solver – which is obviously feasible – is placed slightly inside the unfeasible region, which can be misleading. Secondly, we noticed that due to the inherently imperfect mapping between nD and 2D spaces, equality constraints that compare against constants might not be satisfied during the evaluation, which will make the drawing of feasible regions fail.

9.6 CONCLUSION

In this chapter we presented OptMap, an image-based visualization technique that allows the visualization of multidimensional functions and optimization problems. OptMap exploits the idea of constructing dense maps of high-dimensional spaces, by using direct and inverse projections to map these spaces to a 2D image space. Suitable choices for the sampling of these spaces, as well as using efficient and well-understood direct and inverse projection implementations, make OptMap scalable to real-world problems. We show that OptMap performs well in different scenarios, such as unconstrained

and constrained optimization, with many examples that demonstrate its genericity and speed. Additionally, we show that OptMap can be used for the visualization of standalone high-dimensional functions, even when these are not part of an optimization problem.

Several future work directions exist. First and foremost, it is interesting to consider using more accurate direct and inverse projections for constructing OptMap. Secondly, we consider using OptMap in concrete applications, and gauging its added-value in helping engineers designing better optimization strategies, as opposed to existing tools-of-the-trade for the same task.

This chapter concludes the presentation of applications for multidimensional projections. In Chapter 10, we conclude the core of the thesis by presenting SSNP, Self-Supervised Network Projection, which is a technique for creating direct and inverse projections with a single training. SSNP is the result of the combination of several ideas of using deep learning for projections, such as those presented in Chapters 5, 6 and 7.

10 SELF-SUPERVISED NETWORK PROJECTION

10.1 INTRODUCTION

In Chapters 5 to 9, we presented several techniques and applications which demonstrate the importance of multidimensional projections as tools for the analysis of high-dimensional data. In this chapter, we present yet another technique called Self-Supervised Network Projection, or SSNP, which is based on a single neural network trained with a dual objective, one of reconstructing the input data, as in a typical autoencoder, and the other of data classification using *pseudo-labels* cheaply computed by a clustering algorithm to introduce neighborhood information, since intuitively, clustering gathers and aggregates fine-grained distance information at a higher level, telling how *groups* of samples are similar to each other (at cluster level), and this aggregated information helps us next in producing projections which reflect this higher-level similarity.

This method aims to *jointly* cover all the following characteristics, which, to our knowledge, is not yet achieved by existing DR methods:

Quality (C1): It provides better cluster separation than standard autoencoders, and close to state-of-the-art DR methods, as measured by well-known metrics in DR literature;

Scalability (C2): It can do inference in linear time in the number of dimensions and observations, allowing us to project datasets of up to a million observations and hundreds of dimensions in a few seconds using commodity GPU hardware;

Ease of use (C3): It produces good results with minimal or no parameter tuning;

Genericity (C4): It can handle any kind of high-dimensional data that can be represented as real-valued vectors;

Stability and out-of-sample support (C5): It can project new observations for a learned projection without recomputing it, in contrast to standard t-SNE and any other non-parametric methods;

Inverse mapping (C6): It provides, out-of-the-box, an inverse mapping from the low- to the high-dimensional space;

Clustering (C7): It provides, out-of-the-box, the ability to learn how to mimic clustering algorithms, by assigning labels to unseen data.

We structure this chapter as follows: Section 10.2 discusses related work on dimensionality reduction, Section 10.3 details our method, Section 10.4 presents the results

This chapter is based on publication [53]

that support our contributions outlined above, Section 10.5 discusses our proposal, and Section 10.6 concludes the chapter.

10.2 BACKGROUND

In Chapter 3 we present a detailed study of different DR techniques, and in Chapters 5 to 7 we discuss the use of neural networks to create multidimensional projections. Please refer to those chapters for details.

10.3 METHOD

As stated in Section 5.2, autoencoders have desirable DR properties (simplicity, speed, out-of-sample and inverse mapping capabilities), but create projections of lower quality than, *e.g.*, t-SNE and UMAP. We believe that the key difference is that autoencoders do not use neighborhood information during training, while t-SNE and UMAP do that. This raises the following questions: *Could autoencoders produce better projections if using neighborhood information?* and, if so, How to inject neighborhood information during autoencoder training? Our technique answers both questions by using a network architecture with a *dual* optimization target. First, we have a *reconstruction* target, exactly as in standard autoencoders; next, we use a *classification* target based on pseudo-labels assigned by some *clustering* algorithm *e.g.* K-means [120], Agglomerative Clustering [99], or any other way of assigning labels, including using "true" ground-truth labels if available.

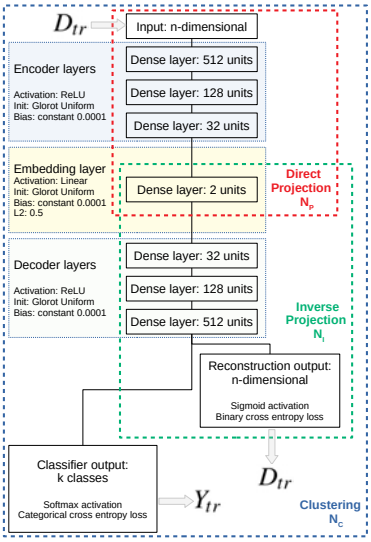


Figure 10.1: SSNP network architecture used for training.

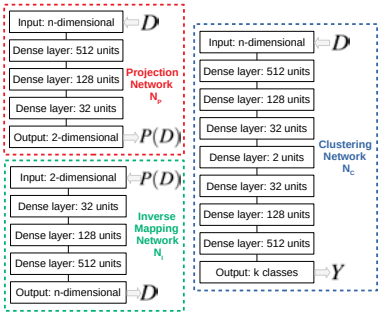


Figure 10.2: Three SSNP networks used during inference.

Our key idea is that labels –ground-truth or given by clustering – are a high-level similarity measure between data samples which can be used to infer neighborhood information, *i.e.*, same-label data are more similar than different-label data. Since classifiers

seek to learn a representation that separates input data based on labels, by adding an extra classifier target to an autoencoder, we learn how to project data with better cluster separation than standard autoencoders.

SSNP first takes a training set $D_{tr} \subset D$ and assigns to it pseudo-labels $Y_{tr} \in \mathbb{Z}$ by using some clustering technique. We then take samples ($\mathbf{x} \in D_{tr}, y \in Y_{tr}$) to train a neural network with two target functions, one for reconstruction, other for classification, which are then added together to form a joint loss. The errors from this joint loss are then back-propagated to the entire network, during training. This network (Figure 10.1) contains a two-unit bottleneck layer, same as an autoencoder, used to generate the 2D projection when in inference mode. After training, we 'split' the trained layers of the network to create three new networks for inference: a *projector* $N_p(\mathbf{x})$, an *inverse projector* $N_i(\mathbf{p})$ and, as a by-product, a *classifier* $N_c(\mathbf{x})$, which mimics clustering algorithm used to create Y_{tr} (see Figure 10.2). The entire training-and-inference operation of SSNP is summarized in Figure 10.3.

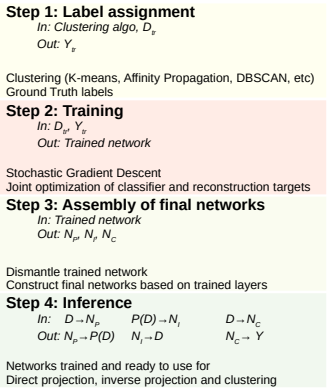


Figure 10.3: SSNP training-and-inference pipeline.

10.4 RESULTS

To evaluate SSNP's performance we propose several experiments to compare it with other DR techniques using different datasets. We select the following evaluation metrics, which are widely used in the projection literature: Trustworthiness (M_t), Continuity (M_c), Neighborhood Hit (M_{nh}) and Shepard diagram correlation (M_s). All metrics are defined in Section 2.2.

We next show how SSNP performs on different datasets when compared to t-SNE, UMAP, autoencoders, and NNP. We use different algorithms to generate pseudo-labels, and also use ground-truth labels. For conciseness, we name SSNP variants using K-means, agglomerative clustering and ground-truth labels as SSNP(Km), SSNP(Agg) and SSNP(GT), respectively. We use two types of datasets: synthetic blobs and real-world data. The synthetic blobs datasets are sampled from a Gaussian distribution where we vary the number of dimensions (100 and 700), the number of cluster centers (5 and 10), and use increasing values of the standard deviation σ. This yields datasets with cluster separation varying from very sharp to fuzzy clusters. All synthetic datasets have 5K samples.

Table 23: Quality measurements for the synthetic blobs experiment with 100 and 700 dimensions, 5 and 10 cluster centers.

		100 dimensions									700 dimensions							
		5 clusters				10 clusters					5 clusters				10 clusters			
Algorithm	σ	M_t	M_c	M_s	M_{nh}	M_t	M_c	M_s	M_{nh}	σ	M_t	M_c	M_s	M_{nh}	M_t	M_c	M_s	M_{nh}
AE	1.3	0.923	0.938	0.547	1.000	0.958	0.963	0.692	1.000	1.6	0.909	0.914	0.739	1.000	0.953	0.955	0.254	1.000
t-SNE		0.937	0.955	0.818	1.000	0.967	0.977	0.192	1.000		0.917	0.951	0.362	1.000	0.960	0.976	0.346	1.000
UMAP		0.921	0.949	0.868	1.000	0.957	0.970	0.721	1.000		0.906	0.933	0.878	1.000	0.954	0.965	0.471	1.000
SSNP-KM		0.910	0.919	0.687	1.000	0.956	0.959	0.602	1.000		0.904	0.908	0.568	1.000	0.953	0.955	0.399	1.000
AE	3.9	0.919	0.926	0.750	1.000	0.959	0.963	0.484	1.000	4.8	0.910	0.914	0.615	1.000	0.953	0.954	0.354	1.000
t-SNE		0.931	0.953	0.707	1.000	0.966	0.978	0.227	1.000		0.914	0.950	0.608	1.000	0.960	0.977	0.331	1.000
UMAP		0.911	0.940	0.741	1.000	0.956	0.969	0.537	1.000		0.906	0.931	0.697	1.000	0.954	0.965	0.390	1.000
SSNP-KM		0.910	0.918	0.622	1.000	0.955	0.958	0.549	1.000		0.905	0.907	0.612	1.000	0.953	0.954	0.296	1.000
AE	9.1	0.905	0.901	0.569	1.000	0.938	0.945	0.328	0.999	11.2	0.911	0.906	0.600	1.000	0.955	0.954	0.382	1.000
t-SNE		0.913	0.951	0.533	1.000	0.948	0.974	0.254	1.000		0.914	0.950	0.492	1.000	0.959	0.977	0.296	1.000
UMAP		0.888	0.939	0.535	1.000	0.929	0.966	0.342	1.000		0.905	0.931	0.557	1.000	0.953	0.965	0.336	1.000
SSNP-KM		0.888	0.917	0.595	0.998	0.927	0.952	0.437	0.995		0.904	0.906	0.557	1.000	0.950	0.945	0.314	0.998

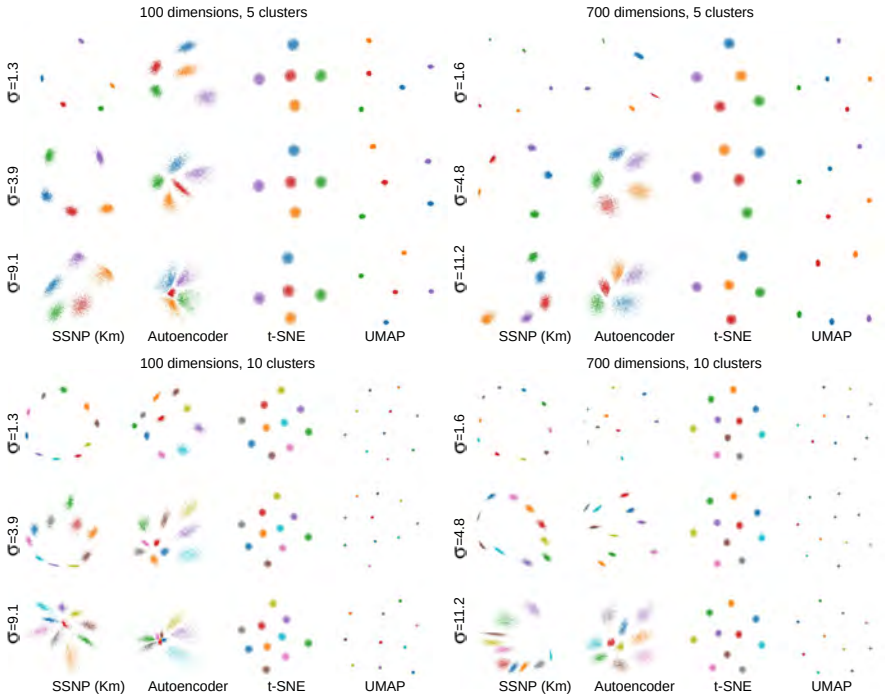


Figure 10.4: Projection of synthetic blobs datasets with SSNP(Km) and other techniques, with different number of dimensions and clusters. In each quadrant, rows show datasets having increasing standard deviation σ.

Real-world datasets are selected from publicly available sources, matching the criteria of being high-dimensional, reasonably large (thousands of samples), and having a non-trivial data structure: MNIST [112], Fashion MNIST [223], Human Activity Recognition (HAR) [8], Reuters Newswire Dataset [204] [176]. All datasets are described in detail in Section 2.3.

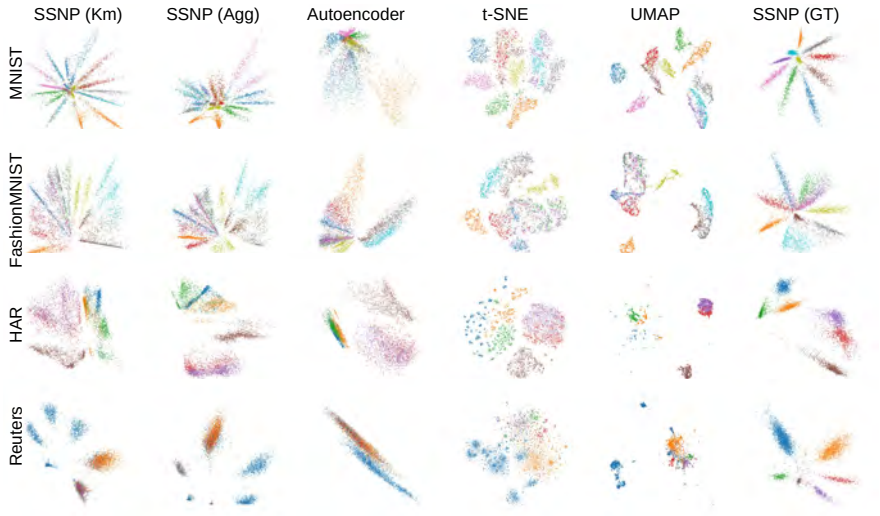


Figure 10.5: Projection of real-world datasets with SSNP and other techniques. Left to right: SSNP using K-means, SSNP using Agglomerative clustering, Autoencoder, t-SNE, UMAP, and SSNP using ground truth labels.

All datasets had their attributes rescaled to the interval $[0, 1]$, to conform with the sigmoid activation function used at the reconstruction layer (see Figure 10.1).

10.4.1 *Quality: Synthetic Datasets*

Figure 10.4 shows the projection of the synthetic blob datasets with SSNP(Km) using the correct number of clusters, alongside Autoencoders, t-SNE and UMAP. We see that in most cases SSNP-Km shows better visual cluster separation than Autoencoders. Also, SSNP-Km preserves the spread of the data better than t-SNE and UMAP, which can be seen by the fact that the projections using these look almost the same regardless of the standard deviation in the data. We omit the plots and measurements for NNP, since these are very close to the ones created by the technique it is trying to mimic, typically t-SNE– see *e.g.* [52].

Table 23 shows the quality measurements for this experiment for the datasets using 5 and 10 cluster centers. We see that SSNP performs very similarly quality-wise to AE, t-SNE, and UMAP. We will bring more insight to this comparison in Section 10.4.2, which studies more challenging, real-world, datasets.

10.4.2 *Quality: Real-World Datasets*

Figure 10.5 shows in the first three columns the projection of real-world datasets by SSNP using pseudo-labels assigned by K-means and Agglomerative Clustering, alongside the projection created by an autoencoder. The next three columns show the same datasets projected by t-SNE, UMAP, and SSNP using ground-truth labels. Again, we omit the plots and measurements for NNP, since they are very close to the ones created by t-SNE and UMAP.

Table 24: Quality measurements for the real-world datasets.

Dataset	Method	M_t	M_c	M_s	M_{nh}
MNIST	SSNP(Km)	0.882	0.903	0.264	0.767
	SSNP(Agg)	0.859	0.925	0.262	0.800
	AE	0.887	0.920	0.009	0.726
	SSNP(GT)	0.774	0.920	0.398	0.986
	NNP	0.948	0.969	0.397	0.891
	t-SNE	0.985	0.972	0.412	0.944
	UMAP	0.958	0.974	0.389	0.913
FashionMNIST	SSNP(Km)	0.958	0.982	0.757	0.739
	SSNP(Agg)	0.950	0.978	0.707	0.753
	AE	0.961	0.977	0.538	0.725
	SSNP(GT)	0.863	0.944	0.466	0.884
	NNP	0.963	0.986	0.679	0.765
	t-SNE	0.990	0.987	0.664	0.843
	UMAP	0.982	0.988	0.633	0.805
HAR	SSNP(Km)	0.932	0.969	0.761	0.811
	SSNP(Agg)	0.926	0.964	0.724	0.846
	AE	0.937	0.970	0.805	0.786
	SSNP(GT)	0.876	0.946	0.746	0.985
	NNP	0.961	0.984	0.592	0.903
	t-SNE	0.992	0.985	0.578	0.969
	UMAP	0.980	0.989	0.737	0.933
Reuters	SSNP(Km)	0.794	0.859	0.605	0.738
	SSNP(Agg)	0.771	0.824	0.507	0.736
	AE	0.747	0.731	0.420	0.685
	SSNP(GT)	0.720	0.810	0.426	0.977
	NNP	0.904	0.957	0.594	0.860
	t-SNE	0.955	0.959	0.588	0.887
	UMAP	0.930	0.963	0.674	0.884

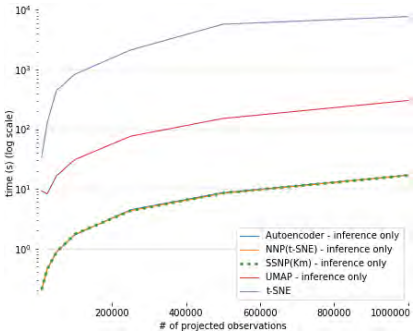
Figure 10.5 shows that SSNP with pseudo-labels shows better cluster separation than Autoencoders, and, for more challenging datasets, such as HAR and Reuters, SSNP with ground-truth labels looks better than t-SNE and UMAP. SSNP and Autoencoder were trained for 10 epochs in all cases. SSNP used twice the number of classes as the target number of clusters for the clustering algorithms used for pseudo-labeling. We see also that SSNP creates elongated clusters, in a star-like pattern. We believe this is due to the fact that one of the targets of the network is a *classifier*, which is trained to partition the space based on the data. This results in placing samples that are near a decision boundary between classes closer to the center of the star pattern; samples that are far away from a decision boundary are placed near the tips of the star, according to its class.

Table 24 shows the quality measurement for this experiment. We see that SSNP with pseudo-labels consistently shows better cluster separation than Autoencoders, as measured by the M_{nh}, as well as better distance correlation, as measured by M_s. For the harder HAR and Reuters datasets, SSNP with ground-truth labels shows M_{nh} results that are competitive and even better than the ones for t-SNE and UMAP. For the M_t and M_c metrics, SSNP outperforms again Autoencoders in most cases; for FashionMNIST and HAR, SSNP yields M_t and M_c values close to the ones for NNP, t-SNE, and UMAP.

Table 25: Set-up time for different methods, using 10K training samples, MNIST dataset. All SSNP variants and Autoencoders were trained for 10 epochs. t-SNE is here for reference only, since it is a non-parametric technique.

Method	Training time (s)
SSNP(GT)	6.029
SSNP(Km)	20.478
SSNP(Agg)	31.954
Autoencoder	3.734
UMAP	25.143
t-SNE	33.620
NNP(t-SNE)	51.181

Figure 10.6: Inference time for SSNP and other techniques. All techniques trained with 10K samples from the MNIST dataset. Inference done on MNIST upsampled up to 1M samples.



10.4.3 *Computational Scalability*

Table 25 shows the time required to set up SSNP and other projection techniques. For SSNP, this means training the network. For t-SNE, this means actual projection of the data, since this technique is non-parametric. All SSNP variants take far under a minute to set up, with SSNP(GT) being the fastest, as it does not need the clustering step. Of the pseudo-labeling varieties, SSNP(Km) is the fastest. We used 10K training samples, which is on the conservative side. In practice, we get good results (quality-wise) with as few as 1K samples.

Figure 10.6 shows the time needed to project up to 1M samples using SSNP and the other techniques. Being GPU-accelerated neural networks, SSNP, Autoencoders and NNP perform very fast, all being able to project up to 1M samples in a few seconds – an order of magnitude faster than UMAP, and over three orders of magnitude faster than t-SNE.

10.4.4 *Inverse Projection*

Figure 27 shows a set of digits from the MNIST dataset – both the actual images x and the ones obtained by inverse projection $P^{-1}(P(\mathbf{x}))$. We see that SSNP(Km) yields results very similar to Autoencoders, so SSNP's dual-optimization target succeeds in learning a good inverse mapping based on the direct mapping given by the pseudo-labels (Section 10.3). Table 26 strengthens this insight by showing the values for Mean Squared Error between original and inversely-projected data for SSNP(Km) and Autoencoder, which, again, are very similar. Furthermore, the SSNP MSE errors are of the same order of magnitude – that is, small – as those obtained by the recent NNInv technique and the older iLAMP [5] one – compare Table 26 with Figure 2 in [59], not included here for space reasons.

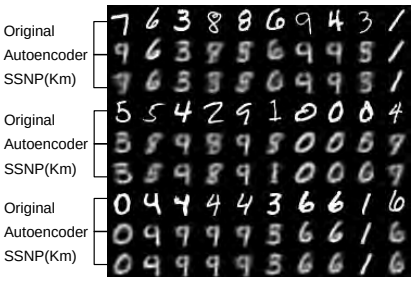
10.4.5 *Data clustering*

Table 28 shows how SSNP performs when doing classification or clustering, which corresponds respectively to its usage of pseudo-labels or ground-truth labels. We see that

Table 26: Inverse projection Mean Square Error (MSE) for SSNP(Km) and Autoencoder, trained with 5K samples, tested with 1K samples.

	SSNP(Km)		Autoencoder	
Dataset	**Train**	**Test**	**Train**	**Test**
MNIST	0.0474	0.0480	0.0424	0.0440
FashionMNIST	0.0309	0.0326	0.0291	0.0305
HAR	0.0072	0.0074	0.0066	0.0067
Reuters	0.0002	0.0002	0.0002	0.0002

Table 27: Sample images from MNIST inversely projected by SSNP and Autoencoder, both trained with 10 epochs and 5K samples. Bright images show the original data.



SSNP generates good results in both cases when compared to the ground-truth labels and, respectively, the underlying clustering algorithm. We stress that this is a side result of SSNP. While one gets this for free, SSNP only *mimics* the underlying clustering algorithm that it learns, rather than doing clustering from scratch.

Table 28: Classification/clustering accuracy of SSNP when compared to ground truth (GT) and clustering labels (Km), trained with 5K observations, test with 1K observations.

	SSNP(GT)		SSNP(Km)	
Dataset	Train	Test	Train	Test
MNIST	0.984	0.942	0.947	0.817
FashionMNIST	0.866	0.815	0.902	0.831
HAR	0.974	0.974	0.931	0.919
Reuters	0.974	0.837	0.998	0.948

10.4.6 *Implementation details*

Table 31 in Section A.3 lists all open-source software libraries used to build SSNP and the other tested techniques. Our neural network implementation leverages the GPU power by using the Keras framework. The t-SNE implementation used is a parallel version of Barnes-Hut t-SNE [125, 207], run on all available CPU cores for all tests. The UMAP reference implementation is not parallel, but is quite fast (compared to t-SNE) and well-optimized. Our implementation, plus all code used in this experiment, are publicly available at `github.com/mespadoto/ssnp`.

10.5 DISCUSSION

We discuss next how SSNP performs with respect to the seven criteria laid out in Section 10.1.

Quality (C1): As shown in Figures 10.4 and 10.5, SSNP provides better cluster separation than Autoencoders, as measured by the selected metrics (Tables 23 and 24). The choice of clustering algorithm does not seem to be a key factor, with K-means and

Agglomerative clustering yielding similar results for all metrics;

Scalability (C2): SSNP, while not as fast to train as standard Autoencoders, is still very fast, with most of the training time being used by clustering – visible by the fact that SSNP(GT)'s training time is close to Autoencoder's one. In our experiments, K-means seems to be faster than Agglomerative clustering, being thus more suitable when training SSNP with very large datasets. Inference time for SSNP is very close to Autoencoders and NNP, and much faster than UMAP (let alone t-SNE), which shows SSNP's suitability to cases where one needs to project large amounts of data, such as streaming applications;

Ease of use (C3): SSNP yielded good projection results with little training (10 epochs), little training data (5K samples) and a simple heuristic of setting the number of clusters for the clustering step to twice the number of expected clusters in the data. Other clustering techniques which do not require setting the number of clusters can be used, such as DBSCAN [61] and Affinity Propagation [69], making SSNP usage even simpler. We see this experiments as part of future work;

Genericity (C4): We show results for SSNP with different types of high-dimensional data, namely tabular (HAR), images (MNIST, FashionMNIST), and text (Reuters). We believe this shows the versatility of the method;

Stability and out-of-sample support (C5): All measurements we show for SSNP are based on inference, *i.e.,* we pass the data through the trained network to compute them. This is evidence of the out-of-sample capability, which allows one to project new data without recomputing the projection, as is the case for t-SNE and other non-parametric methods;

Inverse mapping (C6): SSNP shows inverse mapping results which are, quality-wise, very similar to results from Autoencoders, NNInv and iLAMP, which is evidence of SSNP's inverse mapping abilities;

Clustering (C7): SSNP is able to mimic the behavior of the clustering algorithm used as its input, as a byproduct of the training with labels. We show that SSNP produces competitive results when compared to pseudo- or ground truth labels. Although SSNP is not a clustering algorithm, it provides this for free (with no additional execution cost), which can be useful in cases where one wants to do both clustering and DR.

In addition to the good performance shown for the aforementioned criteria, a key strength of SSNP is its ability of performing all the operations described in Section 10.4 after a *single training phase*, which saves effort and time in cases where all or a subset of those results (*e.g.,* direct projection, inverse projection, clustering) are needed.

10.6 CONCLUSION

We presented a new dimensionality reduction technique called SSNP. Through several experiments, we showed that SSNP creates projections of high-dimensional data that obtain a better visual cluster separation than autoencoders, the technique that SSNP is closest to, and have similar (albeit slightly lower) quality to those obtained by state-of-the-art methods. SSNP is, to our knowledge, the only technique that jointly addresses *all* characteristics listed in Section 10.1, namely producing projections that exhibit a good

visual separation of similar samples, handling datasets of millions of elements in seconds, being easy to use (no complex parameters to set), handling generically any type of high-dimensional data, providing out-of-sample support, and providing an inverse projection function.

One low-hanging fruit is to study SSNP in conjunction with more advanced clustering algorithms than the K-means and agglomerative clustering used in this paper, yielding potentially even better visual cluster separation. A more ambitious, but realizable, goal is to have SSNP learn its pseudo-labeling during training and therefore remove the need for using a separate clustering algorithm. We plan to explore this directions in future work.

11 CONCLUSIONS

In the previous chapters, we gradually developed a body of work with the goal of answering the two research questions outlined in Section 1.2. In this chapter, we conclude this thesis by summarizing how our work managed to answer these questions. We state the contributions made in each chapter in the context of the research questions, and lastly, we outline some directions for future work.

We repeat below the research questions in Section 1.2 for convenience:

RQ1: *How do current projection methods fare concerning quality, stability, and ease of use? What can we teach the practitioner concerning the plethora of such methods available nowadays?*

RQ2: *How can we create projection algorithms which can handle millions of observations with hundreds of dimensions at interactive rates; project new, unseen data, in the same way old data was projected; and tell users what data points belong to the empty spaces between a projection's scatterplot points?*

We addressed question RQ1 by designing, running, and analyzing the results of the benchmark described in Chapter 3. This is the largest benchmark of DR techniques that we are aware of – it evaluates 44 DR techniques over 18 real-world datasets, using grid-search to explore different combinations of values for the techniques' hyperparameters. This resulted in more than 5000 unique runs of each technique. We used 7 different metrics to assess the quality for each run, and to determine which were the best DR techniques and their parameters. The results systematically show that techniques which are local and nonlinear, such as t-SNE and UMAP, perform the best overall. We believe this to be good evidence of the power of nonlinear techniques, and of the need to use local neighborhood information to create good visualizations of complex data. Finally, we made all benchmark code and results public, so others can benefit from the infrastructure developed by us, and even use it to create their own benchmarks and/or compare their DR techniques with existing ones. To our knowledge, ours is the *only* benchmark in DR literature that provides all its results – data, code, methodology – fully openly, thereby allowing replicability, extensibility, and helping usability.

In Chapter 4, we described in detail the *architecture* developed for the benchmark. Such details – how to set up a large-scale quantitative study in terms of software architecture – are unfortunately rare in DR literature. Hence, we see our contribution as a valuable starting point to other researchers that wish to develop benchmarks like ours.

To answer question RQ2, we had first to ask ourselves: How can we have the desirable properties of local and nonlinear techniques, such as good visual cluster separation, while adding fast out-of-sample capability, and without making the process overly complex? The first attempt to answer this question is presented in Chapter 5, where we present our Neural Network Projection (NNP). We show that NNP can mimic the results of many other DR techniques (all techniques that we tested were successfully imitated by NNP) by taking a sample of data projected with some technique of choice and train-

ing a regression neural network in a supervised way. Although the idea is very simple, we show that it is very powerful: NNP can reproduce results obtained with other techniques with a very small error, while being a few orders of magnitude faster when doing inference for new points, thanks to GPU acceleration.

Despite the fact that NNP performs well in a variety of scenarios, we found that the resulting projections typically had a certain fuzziness as compared to the training ground-truth, with some visible scattering of projected points that can be undesirable for some applications. To address this issue, we adopted two approaches, which are described in Chapter 6. First, we did a systematic exploration of the design space for the neural network used in NNP. This resulted in finding optimal settings that improved over the initial implementation of NNP presented in Chapter 5. Next, we changed the architecture of NNP's neural network to be able to accept entire *neighborhoods*, rather than individual samples, during the training of the network. This approach, which we called KNNP, for K-Nearest-Neighbor Projection, led to further improvements in the quality of NNP, with the trade-off of being slightly slower than NNP.

During the course of the development of this thesis, we were faced with *inverse* projections – techniques that map points from low- to high-dimensional spaces – at various stages. First, during our survey work in Chapter 3, we realized that inverse projections are far less studied (and available) than direct projections. Secondly, during the development and testing of NNP (Chapter 5), the natural question arose whether deep learning can also be used for inverse projection tasks. Following this, we sought out to adapt NNP to create inverse projections. This resulted in NNInv, which is presented in Chapter 7. Similar to NNP, NNInv obtains good results, is generic, fast, and has built-in out-of-sample capability, something far more critical for inverse than for direct projections.

In Chapter 8 we present a technique called Decision Boundary Maps. This technique uses direct *and* inverse projections to draw 2D dense maps of ML classifier decision boundaries. We show many variants of this technique, and how it can be useful to understand the inner workings of machine learning (ML) classifiers. This technique relies heavily on NNInv, which enabled significant speed-ups and interactive working for the Decision Boundary Maps.

Chapter 9 presents OptMap, which is a technique for drawing high-dimensional optimization problems in 2D dense maps. This can be seen as a *continuous* variant of the Decision Boundary Maps described earlier. Although OptMap does not use NNP or NNInv for now, it further demonstrates how direct and inverse projections and dense maps can be combined for a different high-dimensional data exploration problem.

To conclude our answer to question RQ2, Chapter 10 presents a new technique called SSNP, for Self-Supervised Network Projection. This is a slightly different approach to using neural networks for projections than the ones at Chapters 5 to 7. First, we use a *clustering* algorithm to assign *pseudo*-labels to the data. Then we use those pseudo-labels to train a network with a joint optimization target, where we train a classifier alongside an autoencoder. By using the pseudo-labels as a proxy for similarity, we show that SSNP can learn to do direct projections with better cluster separation than traditional autoencoders, and without requiring an initial training projection, like NNP and NNInv do. Another benefit of SSNP is that we get three networks in one training: a *direct* projector, which uses the first half of the autoencoder up to the bottleneck layer; an *inverse* projector, which uses the second half of

the autoencoder, from the bottleneck layer to the output; and a *clustering* network, which can mimic the behavior of the clustering algorithm used for training. All of this is achieved computationally very efficiently, at rates comparable to NNP and NNInv.

In terms of future work, we see many possible directions. In particular, there are several sub-areas related to DR and multidimensional projections that can benefit from our results:

Dynamic projections for time-dependent and/or streaming data: With NNP we propose a solution for projecting new data for a known distribution, but what if the distribution of the data changes with time? Dynamic projections can address this problem [166]. NNP and related techniques could be extended to handle dynamic data.

Graph drawing: There are many techniques for graph drawing/layout, and both DR techniques and neural networks have been used to that [76, 108]. Exploring how our DR techniques can be extended to graph drawing is definitely an interesting option.

Explanation of projections: As projection techniques get more sophisticated, there is a need to explain how they work based on input data. In particular, it is interesting to understand why NNP (and related techniques) work so well, what characterizes the cases when they do not work that well, and use this information to further improve their quality.

Handling of categorical data: Most if not all projection techniques deal with quantitative data, while projecting categorical data is a much less explored problem. Extending direct (and inverse) projections such as NNP to handle such data seems to be a quite low-hanging fruit.

Quality metrics: When developing our benchmark, we faced many difficulties for identifying good metrics, let alone finding implementations. We believe there is a need for better quality metrics for projections, as well as proper usable implementations thereof.

Projection software: Many attempts to create projection applications, frameworks, and libraries were made in the last decades. With a few exceptions, most are not maintained and/or lack the implementation of many interesting DR techniques. The development of a new projection library, with implementations that strive to be high-quality, but also didactic in nature, could make the DR arena interesting for a larger audience.

APPENDIX: IMPLEMENTATION DETAILS

A.1 SURVEY

Table 29 lists the software packages used in Chapter 3.

Table 29: Software components implementing the benchmark.

Package name	Available at	Used for
numpy	www.numpy.org	general computations
scipy	www.scipy.org	(statistics, metrics)
pandas	pandas.pydata.org	
scikit-image	scikit-image.org	
scikit-learn	scikit-learn.org	projections
Multicore TSNE	github.com/DmitryUlyanov/Multicore-TSNE	
umap-learn	github.com/lmcinnes/umap	
keras	keras.io	
DR Toolbox	lvdmaaten.github.io/drtoolbox	
Tapkee	tapkee.lisitsyn.me	
Vispipeline	vicg.icmc.usp.br/vicg/tool/1/projection-explorer-pex	
matplotlib	matplotlib.org	visualization

A.2 NNP

Table 30 lists the software packages used in Chapter 5.

Table 30: Software used for implementing the discussed techniques.

Technique	Available at
t-SNE	github.com/DmitryUlyanov/Multicore-TSNE
UMAP	github.com/lmcinnes/umap
PCA MDS Isomap LLE	scikit-learn.org
Autoencoders Our technique	keras.io (TensorFlow backend)
Parametric t-SNE	lvdmaaten.github.io/tsne
LAMP LSP	vicg.icmc.usp.br/vicg

A.3 SSNP

Table 31 lists the software packages used in Chapter 10. Unless noted otherwise, all experiments were run on a 4-core Intel E3-1240 v6 running at 3.7 GHz with 64 GB RAM and an NVidia GeForce GTX 1070 GPU with 8 GB VRAM, where applicable.

Table 31: Software used for the evaluation.

Technique	Available at
SSNP (our technique) Autoencoder	keras.io (TensorFlow backend)
t-SNE	github.com/DmitryUlyanov/Multicore-t-SNE
UMAP	github.com/lmcinnes/umap
K-means Agglomerative Clustering	scikit-learn.org

A.4 CLASSIFIER DECISION BOUNDARY MAPS

Table 32 lists the software packages used in Chapter 8.

Table 32: Software used for implementing the discussed techniques.

Technique	Available at
Logistic Regression Classifier Random Forest Classifier k-Nearest Neighbors Classifier	scikit-learn.org
Convolutional Neural Network	keras.io (TensorFlow backend)

A.5 OPTMAP

Table 33 lists the software packages used in Chapter 9.

Table 33: Software used for the evaluation.

Library	Available at
Images	github.com/JuliaImages/Images.jl
ColorTypes	github.com/JuliaGraphics/ColorTypes.jl
ColorSchemes	github.com/JuliaGraphics/ColorSchemes.jl
Luxor	github.com/JuliaGraphics/Luxor.jl
CSV	github.com/JuliaData/CSV.jl
DataFrames	github.com/JuliaData/DataFrames.jl
MultivariateStats	github.com/JuliaStats/MultivariateStats.jl
Optim	github.com/JuliaNLSolvers/Optim.jl
Clp	github.com/jump-dev/Clp.jl
Cbc	github.com/jump-dev/Cbc.jl
GLPK	github.com/jump-dev/GLPK.jl
Ipopt	github.com/jump-dev/Ipopt.jl

APPENDIX: HYPERPARAMETER SEARCH SPACE

B

B.1 SURVEY

Table 34 presents the hyperparameter search space used for the DR technique evaluated in Chapter 3. Only techniques which have free parameters are listed. To make reading easier, Table 34 (second column) lists the parameter names as used in the respective implementations.

Table 34: Hyperparameter search space used for all projection techniques.

Projection	Parameters π_i	Parameters description and function	Parameter sampling
AE	π_1 : model size	Size of the Autoencoder network	Small (1 hidden layer 2 units), Medium (3 hidden layers 16, 2 and 16 units), Large (f5 hidden layers, 128, 32, 2, 32, 128 units).
DM	π_1 : t	Number of iterations	2, 5, 10
	π_2 : width	Standard deviation for Gaussian kernel	1, 5, 10
GDA	π_1 : kernel	Kernel type	gauss, linear
GPLVM	π_1 : sigma	Standard deviation for Gaussian kernel	0.5, 1.0, 2.0
FA	π_1 :	Max number of iterations	1000, 2000
F-MAP	No parameters		
F-ICA	π_1 : fun	Function used in the approximation to neg-entropy	logcosh, exp
	π_2 : max iter	Max number of iterations	200, 400
IDMAP	π_1 : fraction delta	Fraction of the difference between projected distance and original space distance	2.0, 8.0, 12.0
	π_2 : iter	Number of iterations	100, 200
	π_3 : init	Initialization type	fastmap, random
ISO	π_1 : #neighbors	Number of neighbors to consider for each point	3, 5, 7
L-ISO	π_1 : #neighbors	Number of neighbors to consider for each point	4, 8, 16
LAMP	π_1 : fraction delta	Fraction of the difference between projected distance and original space distance	2.0, 8.0, 12.0
	π_2 : iter	Number of iterations	100, 200
	π_3 : sample type	Sample Type	random, clustering centroid
LE	No parameters		
LLC	π_1 : k	Number of neighbors to consider for each point	8, 12
	π_2 : #analyzers	Number of analyzers	10, 20
	π_3 : max iter	Max number of iterations	200, 400
LLE	π_1 : #neighbors	Number of neighbors to consider for each point	5, 7, 11
	π_2 : max iter	Max number of iterations	100, 200
	π_3 : reg	Regularization constant	0.001, 0.01, 0.1
H-LLE	π_1 : #neighbors	Number of neighbors to consider for each point	7, 11
	π_2 : max iter	Max number of iterations	100, 200
	π_3 : reg	Regularization constant	0.001, 0.01, 0.1
M-LLE	π_1 : #neighbors	Number of neighbors to consider for each point	5, 7, 11
	π_2 : max iter	Max number of iterations	100, 200
	π_3 : reg	Regularization constant	0.001, 0.01, 0.1
LMNN	π_1 : k	Number of neighbors to consider for each point	3, 5, 7
LPP	π_1 : #neighbors	Number of neighbors to consider for each point	4, 7, 11

Table 34 continued

Projection	Parameters π_i	Parameters description and function	Parameter sampling
LSP	π_1 : fraction delta	Fraction of the difference between projected distance and original space distance	2.0, 8.0, 12.0
	π_2 : #iter	Number of iterations	100, 200
	π_3 : #neighbors	Number of neighbors to consider for each point	4, 8, 16
	π_4 : control point type	Method for selecting control points	random, k-means
LTSA	π_1 : #neighbors	Number of neighbors to consider for each point	5, 7, 11
	π_2 : max iter	Max number of iterations	100, 200
	π_3 : reg	Regularization constant	0.001, 0.01, 0.1
L-LTSA	π_1 : #neighbors	Number of neighbors to consider for each point	4, 7, 11
MC	π_1 : analyzers	Number of local factor analyzers	10, 20
	π_2 : max iter	Max number of iterations	200, 400
MCML	No parameters		
MDS	π_1 : init	Number of runs of the SMACOF algorithm	2, 4
	π_2 : max iter	Max number of iterations	300, 500
L-MDS	π_1 : #neighbors	Number of neighbors to consider for each point	4, 7, 11
N-MDS	π_1 : init	Number of runs of the SMACOF algorithm	2, 4
	π_2 : max iter	Max number of iterations	300, 500
L-MVU	π_1 : k1	Number of neighbors to consider for each point	3, 5, 7
	π_2 : k2	Number of neighbors to consider for embedding non-landmark points	8, 12, 15
NMF	π_1 : init	Initialization type	random, nndsvdar
	π_2 : max iter	Max number of iterations	200, 400
	π_3 : alpha	Constant that multiplies the regularization terms	0.0, 0.5
	π_4 : l1 ratio	Regularization mixing constant	0.0, 0.5
PBC	π_1 : init	Initialization type	fastmap, random
	π_2 : cluster factor	Density of clusters in the projected space	1.5, 4.5, 9.0
	π_3 : fraction delta	Fraction of the difference between projected distance and original space distance	2.0, 8.0, 12.0
	π_4 : iter	Number of iterations	100, 200
PCA	No parameters		
I-PCA	No parameters		
K-PCA-P	π_1 : degree	Polynomial degree	2, 3, 5
K-PCA-R	π_1 : gamma	Kernel coefficient	None, 0.05, 0.5
K-PCA-S	π_1 : gamma	Kernel coefficient	None, 0.05, 0.5
P-PCA	π_1 : max iter	Max number of iterations	200, 400
S-PCA	π_1 : alpha	Sparsity controlling parameter	0.01, 0.1, 0.5
	π_2 : max iter	Max number of iterations	1000, 2000
	π_3 : ridge alpha	Amount of ridge shrinkage	0.05, 0.5
PLSP	No parameters		
G-RP	No parameters		
S-RP	No parameters		
R-SAM	No parameters		
T-SNE	π_1 : perplexity	Similar to the number of neighbors	5.0, 15.0, 30.0, 50.0
	π_2 : early exaggeration	Distance between clusters in the projected space	6.0, 12.0, 18.0
	π_3 : #iter	Number of iterations	1000, 3000
SPE	π_1 : #neighbors	Number of neighbors to consider for each point	6, 12, 18
	π_2 : updates	Number of updates	20, 70
T-SVD	π_1 : #iter	Number of iterations	5, 10
UMAP	π_1 : init	Initialization type	spectral, random
	π_2 : min dist	Min distance allowed between points	0.001, 0.01, 0.1, 0.5
	π_3 : #neighbors	Number of neighbors to consider for each point	5, 10, 15

B.2 NNP

As described at various places in Chapter 5, our method learns the style of a given projection technique. For this, suitable training projections are needed. To obtain these, we

used different datasets, and different parameter settings of the underlying projection techniques (to obtain good quality ground-truth projections that we next want to train on). To support replication, Table 35 lists all the hyperparameter settings used for all projection techniques mentioned in Chapter 5, for each considered dataset.

Table 35: Hyperparameters used by all projection techniques and datasets in this chapter.

Dataset	Projection	Hyperparameters
MNIST	t-SNE	perplexity=25, iter=3000
	pt-SNE	perplexity=30
	UMAP	neighbors=10, min_dist=0.001
FashionMNIST	t-SNE	perplexity=10, iter=1000
	UMAP	neighbors=5, min_dist=0.3
Dogs vs Cats	t-SNE	perplexity=30, iter=1000
	UMAP	neighbors=10, min_dist=0.3
IMDB	t-SNE	perplexity=300, iter=5000
	UMAP	neighbors=60, min_dist=0.7
WBC	t-SNE	perplexity=150, iter=3000
	UMAP	neighbors=40, min_dist=0.5
HAR	t-SNE	perplexity=30, iter=3000
	UMAP	neighbors=10, min_dist=0.001
Spambase	t-SNE	perplexity=100, iter=3000
	UMAP	neighbors=30, min_dist=0.1
Seismic Bumps	t-SNE	perplexity=200, iter=3000
	UMAP	neighbors=60, min_dist=0.5
CIFAR100	t-SNE	perplexity=30, iter=5000
	UMAP	neighbors=11, min_dist=0.0
CIFAR100	t-SNE	perplexity=30, iter=5000
	UMAP	neighbors=11, min_dist=0.0
All datasets	Isomap	None
	PCA	None
	Autoencoder 1	1 hidden layer: 2 units(linear)
	Autoencoder 3	3 hidden layers: 16 units(sigmoid),
		2 units(linear), 16 units(sigmoid)
	LLE	neighbors=50
	LSP	fraction delta=8.0, iterations=100
		neighbors=8, control point type=random
	LAMP	fraction delta=8.0, iterations=100
		sample type=random
	MDS	metric=True

C

APPENDIX: MEASUREMENTS FROM EXPERIMENTS

C.1 NNP

Table 36 presents values for all metrics, for all experiments done in Chapter 5, comparing our method to the original techniques.

Table 36: Metrics computed for all datasets and techniques. Superscript *o* indicates "original" technique, superscript *n* indicates our technique.

Dataset	Technique	M^o_{nh}	M^o_t	M^o_c	M^o_s	M^o_{sigma}	M^n_{nh}	M^n_t	M^n_c	M^n_s	M^n_σ
MNIST (2-class)	LAMP	0.996	0.899	0.973	0.813	0.914	0.996	0.901	0.973	0.793	0.916
	LSP	0.995	0.922	0.971	0.787	0.911	0.996	0.916	0.969	0.788	0.913
	PCA	0.997	0.900	0.967	0.771	0.000	0.995	0.898	0.966	0.755	0.000
	UMAP	0.999	0.977	0.987	0.615	1.273	0.999	0.945	0.979	0.609	1.288
	Isomap	0.999	0.903	0.974	0.739	0.000	0.999	0.903	0.972	0.735	0.000
	LLE	0.998	0.931	0.972	0.616	0.003	0.997	0.925	0.967	0.603	0.064
	AE3	0.995	0.948	0.968	0.531	0.386	0.992	0.939	0.969	0.528	0.131
	AE1	0.994	0.891	0.958	0.602	0.378	0.992	0.887	0.959	0.592	0.158
	t-SNE	0.999	0.993	0.985	0.544	1.571	0.997	0.952	0.981	0.547	2.510
	MDS	0.996	0.903	0.944	0.864	0.913	0.990	0.901	0.968	0.810	0.923
Fashion (2-class)	LAMP	0.992	0.914	0.969	0.925	0.922	0.994	0.915	0.968	0.927	0.922
	LSP	0.984	0.911	0.931	0.815	0.926	0.980	0.909	0.953	0.874	0.925
	PCA	0.993	0.909	0.966	0.926	0.000	0.994	0.910	0.966	0.929	0.000
	UMAP	0.998	0.961	0.982	0.796	2.048	0.999	0.931	0.976	0.803	2.077
	Isomap	0.998	0.932	0.970	0.858	0.000	0.999	0.935	0.972	0.867	0.000
	LLE	0.997	0.927	0.952	0.844	0.005	0.998	0.925	0.951	0.850	0.071
	AE3	0.998	0.948	0.958	0.739	0.626	0.999	0.943	0.958	0.734	0.333
	AE1	0.995	0.915	0.963	0.823	0.547	0.994	0.914	0.963	0.823	0.360
	t-SNE	0.998	0.984	0.978	0.621	1.999	0.993	0.926	0.975	0.627	3.356
	MDS	0.993	0.915	0.947	0.937	0.925	0.996	0.918	0.967	0.920	0.930
MNIST	LAMP	0.445	0.710	0.916	0.442	0.956	0.442	0.710	0.917	0.432	0.957
	LSP	0.554	0.761	0.917	0.422	0.957	0.496	0.724	0.897	0.429	0.963
	PCA	0.470	0.732	0.926	0.477	0.000	0.474	0.737	0.926	0.467	0.000
	UMAP	0.894	0.967	0.968	0.308	1.248	0.797	0.846	0.945	0.336	1.260
	Isomap	0.512	0.779	0.962	0.504	0.000	0.508	0.782	0.949	0.505	0.000
	LLE	0.366	0.653	0.841	-0.132	0.008	0.352	0.657	0.834	-0.101	0.221
	AE3	0.606	0.867	0.927	0.165	0.320	0.583	0.846	0.927	0.169	0.201
	AE1	0.449	0.723	0.914	0.337	0.357	0.443	0.725	0.916	0.344	0.227
	t-SNE	0.910	0.984	0.969	0.414	1.642	0.776	0.890	0.959	0.422	2.739
	MDS	0.453	0.786	0.881	0.622	0.926	0.428	0.768	0.920	0.584	0.930
Fashion	LAMP	0.525	0.911	0.976	0.835	0.933	0.523	0.911	0.977	0.842	0.933
	LSP	0.501	0.832	0.863	0.380	0.954	0.476	0.820	0.891	0.608	0.965
	PCA	0.539	0.911	0.975	0.868	0.000	0.528	0.911	0.975	0.872	0.000
	UMAP	0.737	0.978	0.987	0.656	2.006	0.671	0.950	0.982	0.664	2.023
	Isomap	0.608	0.930	0.979	0.754	0.000	0.597	0.929	0.979	0.764	0.000
	LLE	0.623	0.918	0.975	0.579	0.004	0.591	0.906	0.973	0.588	0.071
	AE3	0.648	0.954	0.973	0.530	0.703	0.608	0.940	0.974	0.528	0.341
	AE1	0.535	0.907	0.972	0.710	0.630	0.527	0.905	0.972	0.724	0.376
	t-SNE	0.778	0.990	0.984	0.692	1.736	0.663	0.952	0.981	0.710	3.009
	MDS	0.560	0.921	0.951	0.887	0.927	0.518	0.911	0.973	0.826	0.933

Table 36 continued

Dataset	Technique	M^o_{nh}	M^o_t	M^o_c	M^o_s	M^o_{sigma}	M^n_{nh}	M^n_t	M^n_c	M^n_s	M^n_σ
Dogs vs Cats	LAMP	0.874	0.672	0.886	0.369	0.955	0.879	0.676	0.888	0.403	0.955
	LSP	0.854	0.668	0.795	0.253	0.903	0.869	0.668	0.817	0.326	0.928
	PCA	0.889	0.684	0.884	0.336	0.000	0.891	0.688	0.890	0.339	0.000
	UMAP	0.897	0.814	0.944	0.259	2.232	0.898	0.755	0.920	0.264	2.306
	Isomap	0.869	0.704	0.926	0.415	0.000	0.890	0.723	0.923	0.432	0.000
	LLE	0.887	0.679	0.878	0.175	0.023	0.500	0.562	0.505	0.193	0.000
	AE3	0.881	0.696	0.881	0.231	0.660	0.888	0.698	0.885	0.238	0.654
	AE1	0.886	0.676	0.871	0.249	0.680	0.892	0.682	0.875	0.265	0.655
	t-SNE	0.901	0.914	0.923	0.289	2.446	0.902	0.754	0.910	0.327	3.353
	MDS	0.885	0.651	0.803	0.611	0.896	0.884	0.679	0.857	0.347	0.929
IMDB	LAMP	0.576	0.651	0.604	0.143	0.836	0.584	0.650	0.597	0.137	0.842
	LSP	0.593	0.641	0.714	0.178	0.621	0.590	0.656	0.649	0.179	0.730
	PCA	0.591	0.686	0.646	0.167	0.000	0.591	0.686	0.646	0.166	0.000
	UMAP	0.618	0.690	0.835	0.255	3.615	0.618	0.662	0.734	0.248	3.833
	Isomap	0.588	0.616	0.756	0.242	0.000	0.593	0.638	0.702	0.246	0.000
	LLE	0.584	0.604	0.619	-0.033	0.082	0.569	0.601	0.587	0.003	0.483
	AE3	0.570	0.549	0.604	0.141	0.082	0.570	0.550	0.602	0.117	0.055
	AE1	0.670	0.568	0.654	0.080	0.394	0.669	0.564	0.646	0.053	0.054
	t-SNE	0.640	0.767	0.844	0.510	1.951	0.632	0.667	0.777	0.458	2.155
	MDS	0.575	0.602	0.634	0.333	0.435	0.575	0.594	0.601	0.110	0.796
WBC	LAMP	0.942	0.888	0.978	0.913	0.587	0.937	0.886	0.977	0.912	0.589
	LSP	0.958	0.935	0.979	0.806	0.899	0.914	0.871	0.959	0.833	0.904
	PCA	0.941	0.891	0.979	0.919	0.000	0.936	0.890	0.978	0.919	0.000
	UMAP	0.978	0.966	0.990	0.854	3.890	0.953	0.914	0.976	0.861	3.925
	Isomap	0.900	0.828	0.968	0.520	0.000	0.896	0.814	0.950	0.584	0.000
	LLE	0.818	0.728	0.888	0.685	0.000	0.810	0.736	0.898	0.696	0.239
	AE3	0.939	0.894	0.975	0.793	0.563	0.939	0.895	0.973	0.785	0.480
	AE1	0.933	0.888	0.974	0.863	0.659	0.932	0.888	0.972	0.856	0.482
	t-SNE	0.993	0.995	0.983	0.726	1.113	0.956	0.922	0.970	0.748	1.650
	MDS	0.939	0.898	0.950	0.925	0.546	0.932	0.888	0.976	0.893	0.562
HAR	LAMP	0.820	0.953	0.992	0.959	0.884	0.773	0.941	0.991	0.958	0.884
	LSP	0.937	0.972	0.987	0.707	0.996	0.440	0.692	0.706	0.233	0.996
	PCA	0.725	0.930	0.989	0.941	0.000	0.711	0.927	0.988	0.941	0.000
	UMAP	0.971	0.972	0.965	0.268	1.342	0.538	0.734	0.941	0.328	1.436
	Isomap	0.441	0.609	0.895	-0.093	0.000	0.407	0.632	0.677	-0.075	0.000
	LLE	0.767	0.920	0.984	0.728	0.001	0.665	0.861	0.898	0.746	0.029
	AE3	0.820	0.957	0.986	0.712	0.489	0.804	0.953	0.987	0.717	0.378
	AE1	0.759	0.939	0.989	0.821	0.720	0.784	0.945	0.991	0.820	0.383
	t-SNE	0.990	0.999	0.998	0.641	0.549	0.915	0.977	0.995	0.642	1.607
	MDS	0.798	0.942	0.902	0.927	0.888	0.725	0.922	0.987	0.951	0.904
Spambase	LAMP	0.708	0.635	0.800	0.516	0.771	0.699	0.634	0.798	0.527	0.783
	LSP	0.703	0.626	0.714	0.284	0.798	0.646	0.563	0.644	0.294	0.823
	PCA	0.737	0.639	0.814	0.465	0.000	0.737	0.645	0.816	0.462	0.000
	UMAP	0.779	0.819	0.894	0.583	3.184	0.770	0.714	0.854	0.576	3.258
	Isomap	0.718	0.648	0.807	0.447	0.000	0.724	0.644	0.803	0.488	0.000
	LLE	0.625	0.572	0.726	0.496	0.000	0.611	0.559	0.693	0.511	0.391
	AE3	0.655	0.559	0.637	0.057	0.068	0.647	0.555	0.631	0.066	0.057
	AE1	0.676	0.577	0.700	0.294	0.117	0.672	0.571	0.689	0.288	0.076
	t-SNE	0.800	0.886	0.891	0.581	2.410	0.765	0.725	0.863	0.600	2.880
	MDS	0.738	0.642	0.795	0.707	0.593	0.746	0.655	0.834	0.489	0.727

Table 36 continued

Dataset	Technique	M^o_{nh}	M^o_t	M^o_c	M^o_s	M^o_{sigma}	M^n_{nh}	M^n_t	M^n_c	M^n_s	M^n_σ
Seismic Bumps	LAMP	0.899	0.943	0.960	0.829	0.559	0.904	0.943	0.961	0.834	0.557
	LSP	0.931	0.957	0.962	0.797	0.685	0.909	0.945	0.970	0.858	0.688
	PCA	0.896	0.913	0.950	0.819	0.000	0.899	0.912	0.949	0.824	0.000
	UMAP	0.941	0.984	0.991	0.404	2.757	0.921	0.977	0.987	0.408	2.737
	Isomap	0.916	0.812	0.932	-0.082	0.000	0.905	0.648	0.813	-0.099	0.000
	LLE	0.914	0.957	0.967	0.336	0.000	0.899	0.921	0.942	0.334	0.024
	AE3	0.896	0.900	0.910	0.449	1.155	0.898	0.898	0.911	0.460	0.137
	AE1	0.896	0.858	0.831	0.542	2.622	0.900	0.862	0.835	0.559	0.191
	t-SNE	0.945	0.988	0.992	0.342	0.468	0.920	0.979	0.988	0.353	0.606
	MDS	0.902	0.945	0.930	0.863	0.607	0.906	0.928	0.962	0.822	0.620

[1] Scikit-learn: Machine learning in python. *JMLR*, 12:2825–2830.

[2] D. K. Agrafiotis. Stochastic proximity embedding. *Journal of Computational Chemistry*, 24(10):1215–1221, 2003.

[3] G. Albuquerque, M. Eisemann, and M. Magnor. Perception-based visual quality measures. In *Proc. IEEE VAST*, pages 11–18, 2011.

[4] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami. Contributions to the study of SMS spam filtering: new collection and results. In *Proc. ACM Symposium on Document Engineering*, pages 259–262, 2011.

[5] E. 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 *Proc. IEEE VAST*, pages 53–62, 2012.

[6] E. Amorim, E. V. Brazil, J. Mena-Chalco, L. Velho, L. G. Nonato, F. Samavati, and M. C. Sousa. Facing the high-dimensions: Inverse projection with radial basis functions. *Computers & Graphics*, 48:35–47, 2015.

[7] R. G. Andrzejak, K. Lehnertz, F. Mormann, C. Rieke, P. David, and C. E. Elger. Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state. *Physical Review E*, 64(6):061907–1–061907–8, 2001.

[8] 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. Intl. Workshop on Ambient Assisted Living*, pages 216–223. Springer, 2012.

[9] M. Aupetit. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing*, 10(7):1304–1330, 2007.

[10] M. Aupetit. Sanity check for class-coloring-based evaluation of dimension reduction techniques. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pages 134–141. ACM, 2014.

[11] M. Aupetit and M. Sedlmair. SepMe: 2002 new visual separation measures. In *Proc. IEEE PacificVis*, 2016.

[12] G. Baudat and F. Anouar. Generalized discriminant analysis using a kernel approach. *Neural computation*, 12(10):2385–2404, 2000.

[13] H. U. Bauer and K. R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on neural networks*, 3(4):570–579, 1992.

[14] F. Beck, M. Burch, S. Diehl, and D. Weiskopf. A taxonomy and survey of dynamic graph visualization. *CGF*, 36(1):133–159, 2017.

[15] M. Becker, J. Lippel, A. Stuhlsatz, and T. Zielke. Robust dimensionality reduction for data visualization with deep neural networks. *Graphical Models*, 108:101060, 2020.

[16] R. Becker, W. Cleveland, and M. Shyu. The visual design and control of trellis display. *Journal of Computational and Graphical Statistics*, 5(2):123–155, 1996.

[17] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems (NIPS)*, pages 585–591, 2002.

[18] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[19] B. Benato, A. Telea, and A. Falcão. Semi-supervised learning with interactive label propagation guided by feature space projections. In *Proc. SIBGRAPI*, pages 144–152, 2018.

[20] Y. Bengio, J. Paiement, P. Vincent, O. Dellaleau, N. L. Roux, and M. Ouimet. Out-of-sample extensions for LLE, isomap, MDS, eigenmaps, and spectral clustering. In *Proc. NIPS*, pages 177–184, 2003.

[21] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. SCIPY*, volume 1, pages 3–10, 2010.

[22] E. Bertini, A. Tatu, and D. Keim. Quality metrics in high-dimensional data visualization: An overview and systematization. *IEEE TVCG*, 17(12):2203–2212, 2011.

[23] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[24] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[25] M. Brand. Charting a manifold. In *Proc. NIPS*, pages 985–992, 2002.

[26] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[27] A. Brooke, D. Kendrick, A. Meeraus, R. Raman, and U. America. The general algebraic modeling system, 1998.

[28] A. Buja, D. Cook, and D. F. Swayne. Interactive high-dimensional data visualization. *Journal of Computational and Graphical Statistics*, 5(1):78–99, 1996.

[29] K. Bunte, M. Biehl, and B. Hammer. A general framework for dimensionality reducing data visualization mapping. *Neural Computation*, 24(3):771–804, 2012.

[30] 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 I3D*, pages 83–90, 2010.

[31] C. Chabot, C. Stolte, and P. Hanrahan. Tableau software. *Tableau Software*, 2003.

[32] L. Chen and A. Buja. Local multidimensional scaling for nonlinear dimension reduction, graph drawing, and proximity analysis. *Journal of the American Statistical Association*, 104(485):209–219, 2009.

[33] Y. Chen, M. Crawford, and J. Ghosh. Improved nonlinear manifold learning for land cover classification via intelligent landmark selection. In *Proc. IEEE IGARSS*, pages 545–548, 2006.

[34] F. Chollet and others. *Keras*. 2015. URL keras.io.

[35] S. Christian, S. Ioffe, V. Vanhoucke, and A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning, 2016. arXiv:1602.07261.

[36] P. M. Ciarelli and E. Oliveira. Agglomeration and elimination of terms for dimensionality reduction. In *Proc. IEEE ISDA*, pages 547–552, 2009.

[37] R. R. Coifman and S. Lafon. Diffusion maps. *Applied and computational harmonic analysis*, 21(1):5–30, 2006.

[38] D. Coimbra, R. Martins, T. Neves, A. Telea, and F. Paulovich. Explaining three-dimensional dimensionality reduction plots. *Information Visualization*, 15(2):154–172, 2016.

[39] C. Cortes and V. Vapnik. Support vector machine. *Machine learning*, 20(3):273–297, 1995.

[40] J. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *JMLR*, 16:2859–2900, 2015.

[41] R. Cutura, S. Holzer, M. Aupetit, and M. Sedlmair. VisCoDeR: A tool for visually comparing dimensionality reduction algorithms. In *Proc. ESANN*. Univ. Catholique Louvain, 2018.

[42] G. B. Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. 1990.

[43] S. Dasgupta. Experiments with random projection. In *Proc. of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 143–151. Morgan Kaufmann, 2000.

[44] T. Davidson, D. Warmsley, M. Macy, and I. Weber. Automated hate speech detection and the problem of offensive language. In *Proc. AAAI ICWSM*, pages 512–515, 2017.

[45] J. Deng, W. Dong, R. Socher, L. J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, pages 248–255, 2009.

[46] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10):5591–5596, 2003.

[47] D. van Driel, X. Zhai, Z. Tian, and A. Telea. Enhanced attribute-based explanations of multidimensional projections. In *Proc. EuroVA*. Eurographics, 2020.

[48] I. Dunning, J. Huchette, and M. Lubin. JuMP: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[49] J. Elson, J. J. Douceur, J. Howell, and J. Saul. Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In *Proc. ACM CCS*, pages 366–374, 2007.

[50] D. Engel, L. Hattenberger, and B. Hamann. A survey of dimension reduction methods for high-dimensional data analysis and visualization. In *Proc. IRTG Workshop*, volume 27, pages 135–149. Schloss Dagstuhl, 2012.

[51] M. Espadoto, A. Falcao, N. Hirata, and A. Telea. Improving neural network-based multidimensional projections. In *Proc. IVAPP*. SCITEPRESS, 2020.

[52] M. Espadoto, N. Hirata, and A. Telea. Deep learning multidimensional projections. *Information Visualization*, 2020. doi.org/10.1177/1473871620909485.

[53] M. Espadoto, N. Hirata, and A. Telea. Self-supervised dimensionality reduction with neural networks and pseudo-labeling. In *Proc. IVAPP*. SCITEPRESS, 2021.

[54] M. Espadoto, F. Rodrigues, N. Hirata, and A. Telea. Optmap: Using dense maps for visualizing multidimensional optimization problems. In *Proc. IVAPP*. SCITEPRESS, 2021.

[55] M. Espadoto, R. M. Martins, A. Kerren, N. S. Hirata, and A. C. Telea. Towards a quantitative survey of dimension reduction techniques - companion site, 2019. URL mespadoto.github.io/proj-quant-eval/.

[56] M. Espadoto, R. M. Martins, A. Kerren, N. S. Hirata, and A. C. Telea. Towards a quantitative survey of dimension reduction techniques. *IEEE TVCG*, 2019.

[57] M. Espadoto, R. M. Martins, A. Kerren, N. S. Hirata, and A. C. Telea. Dimensionality reduction online benchmark, 2019. URL github.com/mespadoto/proj-quant-eval.

[58] M. Espadoto, F. C. M. Rodrigues, and A. C. Telea. Visual analytics of multidimensional projections for constructing classifier decision boundary maps. In *Proc. IVAPP*, pages 132–144. SCITEPRESS, 2019.

[59] M. Espadoto, F. C. M. Rodrigues, N. S. T. Hirata, R. Hirata Jr., and A. C. Telea. Deep learning inverse multidimensional projections. In *Proc. EuroVA*. Eurographics, 2019.

[60] M. Espadoto, E. F. Vernier, and A. C. Selecting and sharing multidimensional projection algorithms: A practical view. In *Proc. EuroVis - VisGap Workshop*. The Eurographics Association, 2020.

[61] M. Ester, H. P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. KDD*, pages 226–231, 1996.

[62] R. Fabbri, L. Costa, J. Torellu, and O. Bruno. 2d euclidean distance transform algorithms: A comparative survey. *ACM Computing Survey*, 40(1):1–44, 2008.

[63] C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *ACM SIGMOD*, 24(2):163–174, 1995.

[64] A. Fawzi, S. M. Moosavi-Dezfooli, P. Frossard, and S. Soatto. Empirical study of the topology and geometry of deep networks. In *Proc. IEEE CVPR*, pages 3762–3770, 2018.

[65] I. K. Fodor. A survey of dimension reduction techniques, 2002.

[66] J. Forrest, S. Vigerske, T. Ralphs, L. Hafer, jpfasano, H. G. Santos, M. Saltzman, h-i gassmann, B. Kristjansson, and A. King. coin-or/clp, April 2020.

[67] J. Forrest, S. Vigerske, H. G. Santos, T. Ralphs, L. Hafer, B. Kristjansson, jpfasano, E. Straver, M. Lubin, rlougee, jpgoncal1, h-i gassmann, and M. Saltzman. coin-or/cbc, March 2020.

[68] R. Fourer, D. M. Gay, and B. W. Kernighan. AMPL. A modeling language for mathematical programming, 2003.

[69] B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[70] R. Féraud and F. Clérot. A methodology to explain neural network classification. *Neural Networks*, 15(2):237–246, 2002.

[71] A. Gisbrecht and B. Hammer. Data visualization by nonlinear dimensionality reduction. *WIREs Data Mining Knowledge Discovery*, 5:51–73, 2015.

[72] A. Globerson and S. T. Roweis. Metric learning by collapsing classes. In *Proc. NIPS*, pages 451–458, 2006.

[73] Y. Goldberg and Y. Ritov. Local procrustes for manifold embedding: a measure of embedding quality and embedding algorithms. *Machine learning*, 77(1):1–25, 2009.

[74] J. Goldberger, S. Roweis, G. E. Hinton, and R. R. Salakhutdinov. Neighbourhood components analysis. *NIPS*, 17:513–520, 2005.

[75] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2014. arXiv:1412.6572.

[76] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proc. ACM SIGKDD*, pages 855–864, 2016.

[77] B. Guenin, J. Könemann, and L. Tuncel. *A gentle introduction to optimization*. Cambridge University Press, 2014.

[78] I. Guyon, A. Saffari, G. Dror, and G. Cawley. Agnostic learning vs. prior knowledge challenge. In *Proc. IEEE IJCNN*, pages 829–834, 2007.

[79] W. W. Hager and H. Zhang. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):113–137, 2006.

[80] N. Halko, P. G. Martinsson, and J. A. Tropp. *Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions*. 2009.

[81] L. Hamel. Visualization of support vector machines with unsupervised learning. In *Proc. IEEE CIBCB*, 2006.

[82] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. IEEE ICCV*, pages 1026–1034, 2015.

[83] X. He and P. Niyogi. Locality preserving projections. In *Proc. NIPS*, pages 153–160, 2004.

[84] M. Hearst. What is text mining. *SIMS, UC Berkeley*, 2003.

[85] N. Heulot, J. D. Fekete, and M. Aupetit. *Visualizing Dimensionality Reduction Artifacts: An Evaluation.* 2017.

[86] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[87] G. E. Hinton and S. T. Roweis. Stochastic neighbor embedding. In *Proc. NIPS*, pages 857–864, 2003.

[88] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[89] 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.

[90] M. Hopkins, E. Reeber, G. Forman, and J. Suermondt. *Spambase dataset.* 1999.

[91] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks, 2016. arXiv:1608.06993.

[92] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(3):90–95, 2007.

[93] A. Hyvarinen. Fast ICA for noisy data using gaussian moments. In *Proc. IEEE ISCAS*, volume 5, pages 57–61, 1999.

[94] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multidimensional geometry. In *Proc. IEEE Visualization*, pages 361–378, 1990.

[95] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato. Local affine multidimensional projection. *IEEE TVCG*, 17(12):2563–2571, 2011.

[96] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato. Vispipeline, 2011. URL `vicg.icmc.usp.br/vicg/tool/1/projection-explorer-pex`.

[97] I. T. Jolliffe. Principal component analysis and factor analysis. In *Principal Component Analysis*, pages 115–128. Springer, 1986.

[98] L. V. Kantorovich. Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422, 1960.

[99] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* Wiley, 2005.

[100] J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE TVCG*, 19(3):495–513, 2013.

[101] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2013. arXiv:1312.6114.

[102] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. arXiv:1412.6980.

[103] A. Kirk. *Data Visualization: a successful design process*. Packt Publishing Ltd, 2012.

[104] D. Kotzias, M. Denil, N. d. Freitas, and P. Smyth. From group to individual labels using deep features. In *Proc. ACM SIGKDD*, pages 597–606, 2015.

[105] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, MSc thesis, Dept. of Computer Science, Univ. of Toronto, Canada, 2009.

[106] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[107] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *NIPS*, pages 950–957, 1992.

[108] J. F. Kruiger, P. E. Rauber, R. M. Martins, A. Kerren, S. Kobourov, and A. C. Telea. Graph layouts by t-sne. 36(3):283–294, 2017.

[109] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[110] S. Lafon and A. B. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and dataset parameterization. *IEEE TVCG*, 28(9):1393–1403, 2006.

[111] N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Proc. NIPS*, pages 329–336, 2004.

[112] Y. LeCun and C. Cortes. MNIST handwritten digits dataset, 2010. URL yann.lecun.com/exdb/mnist.

[113] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Proc. NIPS*, pages 556–562, 2001.

[114] J. A. Lee and M. Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7):1431–1443, 2009.

[115] S. Lespinats and M. Aupetit. CheckViz: Sanity check and topological clues for linear and nonlinear mappings. *Computer Graphics Forum*, 30(1):113–125, 2011.

[116] S. Lespinats, M. Aupetit, and A. Meyer-Baese. ClassiMap: A new dimension reduction technique for exploratory data analysis of labeled data. *IJPRAI*, 29(6), 2015.

[117] S. Lisitsyn, C. Widmer, and F. J. I. Garcia. Tapkee: An efficient dimension reduction library. *JMLR*, 14(1):2355–2359, 2013.

[118] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[119] 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.

[120] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[121] W. Lueks, A. Gisbrecht, and B. Hammer. Visualizing the quality of dimensionality reduction. *Neurocomputing*, 112:109–123, 2013.

[122] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proc. of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150. Association for Computational Linguistics, 2011.

[123] L. v. d. Maaten. An introduction to dimensionality reduction using matlab. Technical report, Maastricht University, 2007.

[124] L. v. d. Maaten. Learning a parametric embedding by preserving local structure. In *Proc. AI-STATS*, 2009.

[125] L. v. d. Maaten. Barnes-hut-SNE, 2013. arXiv:1301.3342.

[126] L. v. d. Maaten. Accelerating t-SNE using tree-based algorithms. *JMLR*, 15:3221–3245, 2014.

[127] L. v. d. Maaten and G. Hinton. Visualizing data using t-SNE. *JMLR*, 9:2579–2605, 2008.

[128] L. v. d. Maaten and E. Postma. Dimensionality reduction: A comparative review. Technical report, Tilburg University, Netherlands, 2009.

[129] L. v. d. Maaten, E. Postma, and H. v. d. Herik. MATLAB toolbox for dimensionality reduction. *Maastricht Univ.*, 2007.

[130] A. Makhorin. GLPK: GNU Linear Programming Kit), 2008. URL www.gnu.org/software/glpk/glpk.html.

[131] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, 2016. arXiv:1603.09320.

[132] C. D. Manning, H. Schütze, and P. Raghavan. *Introduction to Information Retrieval*, volume 39. Cambridge University Press, 2008.

[133] R. Martins, D. Coimbra, R. Minghim, and A. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41: 26–42, 2014.

[134] R. M. Martins, R. Minghim, A. C. Telea, and others. Explaining neighborhood preservation for multidimensional projections. In *CGVC*, pages 7–14, 2015.

[135] P. Mattson, C. Cheng, C. Coleman, G. Diamos, P. Micikevicius, D. Patterson, H. Tang, G. Y. Wei, P. Bailis, V. Bittorf, and others. MLPerf benchmark, 2019. arXiv:1910.01500.

[136] M. McCann and A. Johnston. *SECOM dataset*. 2008.

[137] B. McCormick. Visualization in scientific computing. *Computer graphics*, 21(6), 1987.

[138] L. McInnes and J. Healy. UMAP: Uniform manifold approximation and projection for dimension reduction, 2018. arXiv:1802.03426.

[139] J. Melville. SmallVis benchmark, 2018. URL github.com/jlmelville/smallvis.

[140] T. Mendonça, P. M. Ferreira, J. S. Marques, A. R. Marcal, and J. Rozeira. PH 2-a dermoscopic image database for research and benchmarking. In *2013 35th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*, pages 5437–5440. IEEE, 2013.

[141] M. A. Migut, M. Worring, and C. J. Veenman. Visualizing multi-dimensional decision boundaries in 2d. *Data Mining and Knowledge Discovery*, 29(1):273–295, 2015.

[142] R. Minghim, F. V. Paulovich, and A. A. Lopes. Content-based text mapping using multi-dimensional projections for exploration of document collections. In *Proc. SPIE*, volume 6060. Intl. Society for Optics and Photonics, 2006.

[143] S. Moro, P. Cortez, and P. Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[144] R. Motta, R. Minghim, A. Lopes, and M. Oliveira. Graph-based measures to assist user assessment of multidimensional projections. *Neurocomputing*, 150:583–598, 2015.

[145] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[146] S. A. Nene, S. K. Nayar, H. Murase, and others. Columbia object image library (coil-20), 1996.

[147] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *Proc. NIPS workshop on deep learning and unsupervised feature learning*, 2011.

[148] L. Nonato and M. Aupetit. Multidimensional projection for visual analytics: Linking techniques with distortions, tasks, and layout enrichment. *IEEE TVCG*, 2018. DOI 10.1109/TVCG.2018.2846735.

[149] M. OL, S. R, and W. WH. Pattern recognition via linear programming: Theory and application to medical diagnosis. *Large-scale numerical optimization*, pages 22–31, 1990.

[150] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE TKDE*, 22(10):1345–1359, 2010.

[151] Papermill. Papermill. URL papermill.readthedocs.io.

[152] M. Y. Park and T. Hastie. L1-regularization path algorithm for generalized linear models. *J Royal Stat Soc: Series B*, 69(4):659–677, 2007.

[153] F. V. Paulovich and R. Minghim. Text map explorer: a tool to create and explore document maps. In *Proc. Intl. Conference on Information Visualisation (IV)*, pages 245–251. IEEE, 2006.

[154] 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.

[155] F. V. Paulovich, C. T. Silva, and L. G. Nonato. Two-phase mapping for projecting massive datasets. *IEEE TVCG*, 16(6):1281–1290, 2010.

[156] E. Pekalska, D. d. Ridder, R. P. W. Duin, and M. A. Kraaijveld. A new method of generalizing sammon mapping with application to algorithm speed-up. In *Proc. ASCI*, volume 99, pages 221–228, 1999.

[157] J. Peltonen and Z. Lin. Information retrieval approach to meta-visualization. *Machine Learning*, 99(2):189–229, 2015.

[158] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. *Computer Graphics Forum*, 35(3):21–30, 2016.

[159] N. Pezzotti, B. Lelieveldt, L. v. d. Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable t-SNE for progressive visual analytics. *IEEE TVCG*, 23:1739–1752, 2017.

[160] N. Pezzotti, J. Thijssen, A. Mordvintsev, T. Hollt, B. v. Lew, B. Lelieveldt, E. Eisemann, and A. Vilanova. GPGPU linear complexity t-SNE optimization. *IEEE TVCG*, 26(1):1172–1181, 2020.

[161] G. Polzlbauer. Survey and comparison of quality measures for self-organizing maps. In *Proc. Workshop on Data Analysis (WDA)*, pages 67–82, 2004.

[162] K. Potdar, P. Pardawala, and C. Pai. A comparative study of categorical variable encoding techniques for neural network classifiers. *Intl J of Computer Applications*, 175(4):7–9, 2017.

[163] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[164] 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 SIGCHI*, pages 318–322, 1994.

[165] L. A. Rastrigin. Systems of extremal control. *Nauka*, 1974.

[166] P. Rauber, A. X. Falcão, and A. Telea. Visualizing time-dependent data using dynamic t-SNE. In *Proc. EuroVis: Short Papers*, pages 73–77, 2016.

[167] P. E. Rauber, S. G. Fadel, A. X. Falcao, and A. C. Telea. Visualizing the hidden activity of artificial neural networks. *IEEE TVCG*, 23(1):101–110, 2017.

[168] P. E. Rauber, A. X. Falcão, and A. C. Telea. Projections as visual aids for classification system design. *Information Visualization*, 17(4):282–305, 2017.

[169] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proc. ACM SIGMOD KDD*, pages 1135–1144, 2016.

[170] F. Rodrigues, M. Espadoto, R. Hirata, and A. C. Telea. Constructing and visualizing high-quality classifier decision boundary maps. *Information*, 10(9):280, 2019.

[171] F. C. M. Rodrigues, R. Hirata, and A. C. Telea. Image-based visualization of classifier decision boundaries. In *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 353–360. IEEE, 2018.

[172] C. Rohkohl, B. Keck, H. Hofmann, and J. Hornegger. RabbitCT—an open platform for benchmarking 3d cone-beam reconstruction algorithms a. *Medical Physics*, 36 (9):3940–3944, 2009.

[173] H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[174] D. A. Ross, J. Lim, R. S. Lin, and M. H. Yang. Incremental learning for robust visual tracking. *International Journal of Computer Vision*, 77(1):125–141, 2008.

[175] S. T. Roweis and L. L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[176] G. Salton and M. J. McGill. *Introduction to modern information retrieval*. McGraw-Hill, 1986.

[177] F. S. Samaria and A. C. Harter. Parameterisation of a stochastic model for human face identification. In *Proc. IEEE WACV*, pages 138–142, 1994.

[178] H. Sanftmann and D. Weiskopf. 3d scatterplot navigation. *IEEE TVCG*, 18(11): 1969–1978, 2012.

[179] T. Schreck, T. v. Landesberger, and S. Bremm. Techniques for precision-based visual analysis of projected data. *Information Visualization*, 9(3):181–193, 2010.

[180] A. Schulz, A. Gisbrecht, and B. Hammer. Using discriminative dimensionality reduction to visualize classifiers. *Neural Processing Letters*, 42(1):27–54, 2015.

[181] B. Schölkopf, A. Smola, and K. Müller. Kernel principal component analysis. In *Proc. International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[182] B. Schölkopf, A. J. Smola, and K. R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[183] M. Sedlmair and M. Aupetit. Data-driven evaluation of visual quality measures. *Comp Graph Forum*, 34(3):545–559, 2015.

[184] M. Sedlmair, T. Munzner, and M. Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE TVCG*, pages 2634–2643, 2013.

[185] C. Seifert, V. Sabol, and W. Kienreich. Stress maps: analysing local phenomena in dimensionality reduction based visualisations. In *Proc. IEEE VAST*, 2010.

[186] L. Sharan, R. Rosenholtz, and E. Adelson. Material perception: What can you see in a brief glance? *Journal of Vision*, 9(8):784–784, 2009.

[187] M. Sikora and L. Wróbel. Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines. *Archives of Mining Sciences*, 55(1):91–114, 2010.

[188] R. d. Silva, P. Rauber, R. Martins, R. Minghim, and A. C. Telea. Attribute-based visual explanation of multidimensional projections. In *Proc. EuroVA*, 2015.

[189] V. D. Silva and J. B. Tenenbaum. Sparse multidimensional scaling using landmark points, 2004.

[190] M. Sips, B. Neubert, J. Lewis, and P. Hanrahan. Selecting good views of high-dimensional data using class consistency. *Comp Graph Forum*, 28(3):831–838, 2009.

[191] D. Smilkov and S. Carter. *TensorFlow Playground*. 2019.

[192] C. Sorzano, J. Vargas, and A. Pascual-Montano. A survey of dimensionality reduction techniques, 2014. arXiv:1403.2877.

[193] N. Srebro and A. Shraibman. Rank, trace-norm and max-norm. In *Intl. Conf. on Computational Learning Theory*, pages 545–560. Springer, 2005.

[194] M. Styblinski and T. S. Tang. Experiments in nonconvex optimization: stochastic approximation with function smoothing and simulated annealing. *Neural Networks*, 3(4):467–483, 1990.

[195] D. F. Swayne, D. Cook, and A. Buja. XGobi: Interactive dynamic data visualization in the x window system. *J Computational and Graphical Statistics*, 7(1):113–130, 1998.

[196] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proc. CVPR*, pages 2818–2826, 2016.

[197] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2013. arXiv:1312.6199.

[198] J. Tang, J. Liu, M. Zhang, and Q. Mei. Visualizing large-scale and high-dimensional data. In *Proc. WWW*, pages 287–297, 2016.

[199] A. Tatu, G. Albuquerque, M. Eisemann, J. Schneidewind, H. Theisel, M. Magnor, and a. D. Keim. Combining automated analysis and visualization techniques for effective exploration of high dimensional data. In *Proc. IEEE VAST*, pages 59–66, 2009.

[200] 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 *Proc. AVI*, pages 49–56. ACM, 2010.

[201] Y. W. Teh and S. T. Roweis. Automatic alignment of hidden representations. In *Proc. NIPS*, pages 841–848, 2002.

[202] A. C. Telea. Combining extended table lens and treemap techniques for visualizing tabular data. In *Proc. EuroVis*, pages 120–127, 2006.

[203] J. B. Tenenbaum, V. D. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[204] M. Thoma. The Reuters dataset, July 2017. URL martin-thoma.com/nlp-reuters.

[205] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society*, 61(3):611–622, 1999.

[206] W. S. Torgerson. *Theory and Methods of Scaling*. Wiley, 1958.

[207] D. Ulyanov. Multicore-TSNE, 2016. URL github.com/DmitryUlyanov/Multicore-TSNE.

[208] J. Venna and S. Kaski. Visualizing gene interaction graphs with local multidimensional scaling. In *Proc. ESANN*, pages 557–562, 2006.

[209] E. Vernier, R. Garcia, I. d. Silva, J. Comba, and A. Telea. Quantitative evaluation of time-dependent multidimensional projection techniques. In *Proc. EuroVis*, 2020.

[210] E. F. Vernier, J. Comba, and A. Telea. Quantitative comparison of dynamic treemaps for software evolution visualization. In *Proc. IEEE VISSOFT*, 2018.

[211] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[212] S. Walt, S. C. Colbert, and G. Varoquaux. The NumPy array: a structure for efficient numerical computation. *Comp Sci Eng*, 13(2):22–30, 2011.

[213] Y. Wang, K. Feng, X. Chu, J. Zhang, C. W. Fu, M. Sedlmair, X. Yu, and B. Chen. A perception-driven approach to supervised dimensionality reduction for visualization. *IEEE TVCG*, 24(5):1828–1840, 2018.

[214] M. Wattenberg. How to use t-SNE effectively, 2016. URL distill.pub/2016/misread-tsne.

[215] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification. In *Proc. NIPS*, pages 1473–1480, 2006.

[216] K. Q. Weinberger, F. Sha, and L. K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proc. ICML*, 2004.

[217] K. Q. Weinberger, B. Packer, and L. K. Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *AISTATS*. Citeseer, 2005.

[218] H. Wickham. Tidy data. *Journal of Statistical Software*, 59(10), 2014.

[219] R. Wicklin. Visualize the feasible region for a constrained optimization, 2018. URL blogs.sas.com/content/iml/2018/11/07/visualize-feasible-region-constrained-optimization.html.

[220] J. J. van Wijk and R. van Liere. Hyperslice. In *Proc. Visualization*, pages 119–125. IEEE, 1993.

[221] J. J. van Wijk and A. Telea. Enridged contour maps. In *Proc. IEEE Visualization*, pages 69–543, 2001.

[222] A. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. In *NIPS*, pages 4148–4158. Curran Associates, Inc., 2017.

[223] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms, 2017. arXiv:1708.07747.

[224] H. Xie, J. Li, and H. Xue. A survey of dimensionality reduction techniques based on random projection, 2017. arXiv:1706.04371.

[225] Y. Yao, L. Rosasco, and A. Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315, 2007.

[226] A. Yates, A. Webb, M. Sharpnack, H. Chamberlin, K. Huang, and R. Machiraju. Visualizing multidimensional data with glyph SPLOMs. *Computer Graphics Forum*, 33(3):301–310, 2014.

[227] H. Yin. Nonlinear dimensionality reduction and data visualization: A review. *Intl. Journal of Automation and Computing*, 4(3):294–303, 2007.

[228] T. Zhang, J. Yang, D. Zhao, and X. Ge. Linear local tangent space alignment and application to face recognition. *Neurocomputing*, 70(7):1547–1553, 2007.

[229] Z. Zhang and J. Wang. MLLE: Modified locally linear embedding using multiple weights. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1593–1600, 2007.

[230] Z. Zhang and H. Zha. Principal manifolds and nonlinear dimensionality reduction via tangent space alignment. *SIAM Journal on Scientific Computing*, 26(1):313–338, 2004.

[231] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006.

ACKNOWLEDGMENTS

This work would not be possible without the dedication, patience and hard work of my two supervisors, Nina and Alex. My deepest thanks to both of you for putting up with me and my idiosyncrasies over the years. You made my life a lot easier, and I certainly made yours a bit more difficult.

Many thanks goes to friend and colleague Caio Rodrigues, for the scientific, philosophical, and existential discussions over beer and *vlaamse frites. Proost!*.

I also thank prof. Arnaldo Mandel, for showing me that Mark Twain was right: "*It ain't what you don't know that gets you into trouble; it's what you know for sure that just ain't so.*"

Last but certainly not least, I thank my family, Adriana, James, and Robert, for putting up with my long leaves of absence while undertaking this project, and for the love and support over the years. This work is dedicated to you.

This thesis was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) and by the University of Groningen. My sincere thanks to both organizations for their support.

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 LaTeX and LyX:

http://code.google.com/p/classicthesis/

Final Version as of February 20, 2021 (`classicthesis`).