VISUAL ANALYTICS FOR MACHINE LEARNING

FRANCISCO CAIO MAIA RODRIGUES

Computing and Leveraging Decision Boundary Maps

Cover: Decision zones, boundary maps, and distances to boundaries for the classification of the MNIST dataset.

Visual Analytics for Machine Learning
Computing and Leveraging Decision Boundary Maps
Francisco Caio Maia Rodrigues
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

Visual Analytics for Machine Learning

Computing and Leveraging Decision Boundary Maps

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

Wednesday 14 October 2020 at 18.00 hours

by

Francisco Caio Maia Rodrigues

born on May 4th, 1989
in Fortaleza, Brazil

Supervisors
Prof. A. C. Telea
Prof. R. Hirata Jr.

Assessment committee
Prof. M. Biehl
Prof. R. Marcondes Cesar
Prof. A. X. Falcão
Prof. N. Petkov

ABSTRACT

Machine learning classifiers construct decision boundaries that partition data space into a set of regions to which labels are assigned. Understanding these decision boundaries can notably help the actual practical usage of such classifiers (by answering questions such as showing *how* a certain model is expected to behave on an empty region), as well as give insights on how to improve the training of a given model (by answering questions such as telling *where* should more training data be provided). In this thesis we propose and explore visual analytics methods for the explicit creation, construction, and use of decision zones of machine learning classifiers.

Current methods employed to visualize how a classifier behaves on a dataset mainly use color-coded sample scatterplots, which do not explicitly show the actual decision boundaries or confusion zones. We propose an image-based technique to improve such visualizations. The method samples the 2D space of a projection and color-codes relevant classifier outputs, such as the majority class label, the confusion, and the sample density, to create a dense visual depiction of the high-dimensional decision boundaries. Our technique is simple to implement, handles any classifier, and has only two simple-to-control free parameters. We demonstrate our proposal on several real-world high-dimensional datasets, classifiers, direct and inverse projection techniques. To our knowledge, our work is the first that can create such explicit depictions of decision boundaries and decision zones for any dataset and any classifier, without explicit knowledge of the classifier's internals.

Based on these visual depictions of decision boundaries, we developed a visual analytics workflow and associated tooling that allows users to perform two common techniques in machine learning – data augmentation and interactive labeling of unseen samples. We show that our approach can be used to perform guided data augmentation in order to shape the decision boundaries learned by a classifier according to the user's input. For interactive labeling, we show that our proposed visual depiction of decision boundaries helps in producing improved labeling in an active learning scenario.

SAMENVATTING

Machinaal leren classificatoren bouwen beslissingsranden die de dataruimte partitioneren in een verzameling van gebieden waaraan etiketten toegekend kunnen worden. Het begrijpen van deze beslissingsranden kan het praktisch gebruik van classificatoren verhelpen, bijvoorbeeld door het laten zien *hoe* een model zich gedraagt over een leeg datagebied; en kan ook inzichten geven tot het verbeteren van de training van een gegeven model, door het beantwoorden van vragen zoals *waar* meer trainingsgegevens nodig zijn. Dit proefschrift presenteert en exploreert *visual analytics* methoden voor de expliciete creatie, constructie, en gebruik van beslissingsranden van classificatoren in machinaal leren.

Huidige methoden visualiseren hoe een classificator zich gedraagt over een gegevensverzameling gebruiken typisch gekleurde *scatterplots* van data punten; deze laten niet expliciet zien waar de beslissingsranden of confusiegebieden zich bevinden. Wij verbeteren dit door het gebruik van *image-based* visualisatie. De methode meet de 2D ruimte van een projectie en codeert via kleur de relevante resultaten van een classificator, zoals de dominante klasse-etiket, de confusie, en de sampledichtheid. Dit genereert een dichte visuele afbeelding van de hoogdimensionale beslissingsranden. Onze techniek is makkelijk te implementeren, werkt voor alle classificatoren, en heeft maar twee eenvoudig te controleren parameters. Wij demonstreren ons voorstel voor verschillende hoogdimensionale datasets met reële gegevens, evenals voor directe en inverse projectietechnieken. Tot zover wij weten is onze techniek de eerste die dergelijke expliciete afbeeldingen van beslissingsranden en beslissingsgebieden kan creëren voor elk dataset en classificator, zonder kennis van de details van de classificator.

Wij gebruiken de visuele afbeelding van beslissingsranden om een visueel analyse *workflow* en implementatie te ontwikkelen die gebruikers in staat stelt om twee standard operaties in machineleer toe te passen – data augmentatie en interactieve *labeling* van ongeziene samples. Onze techniek kan gebruikt worden om data augmentatie uit te voeren om de beslissingsranden van een classificator vorm te geven volgens de *input* van de gebruiker. Voor interactieve *labeling* laten wij zien dat ons voorgestelde visuele afbeelding van beslissingsranden verhelpt de constructie van verbeterde etiketten in een activeleer scenario.

RESUMO

Modelos de aprendizado de máquina chamados classificadores constroem fronteiras de decisão que particionam um certo espaço de dados em um conjunto de regiões, associando-as a um rótulo. Entender a estrutura e forma de tais fronteiras de decisão pode ser de grande ajuda no uso prático de tais classificadores, respondendo, por exemplo, questões sobre *como* espera-se que certo modelo se comporte em uma região vazia do espaço. Além disso, tal entendimento pode ajudar a dar ideias que levem a melhoria do treino de um certo modelo, por exemplo através da indicação de *onde* mais dados de treino poderiam ser coletados. Nessa tese, propomos e exploramos métodos de visualização para a criação e o uso de modelos visuais das fronteiras de decisão inferidas por classificaores de aprendizado de máquina.

Atualmente, métodos utilizados para visualizar o comportamento de um classificador treinado em um certo conjunto de dados fazem uso *scatterplot*, colorindo os pontos de acordo com a classe atribuida pelo modelo. Nesta tese, propomos uma técnica baseada em imagens para aprimorar tais visualizações. Nosso método amostra o espaço 2D de uma projeção, codificando nas cores dos pixels aspectos relevantes de um classificador treinado, como a maioria dos rótulos naquela região, o grau de confusão e a densidade de amostras, criando uma imagem densa das fronteiras inferidas em espaços de alta dimensão. O método proposto é simples de implementar, funciona para qualquer classificador e possui apenas dois parâmetros intuitivos. Demonstramos o uso da técnica proposta em diferentes *datasets* de alta dimensionalidade, classificadores, projeções diretas e inversas. No nosso conhecimento, nosso trabalho é o primeiro capaz de criar tais visualizações explícitas das fronteiras de classificadores, para qualquer dataset e classificador, sem necessidade do conhecimento do funcionamento de detalhes internos dos modelos.

Baseado nas descrições visuais das fronteiras de decisão, nós desenvolvemos um *workflow* de *visual analytics* e uma ferramenta gráfica que permite aos usuários realizarem a rotulagem interativa de amostras. Mostramos ainda que o nosso método proposto de visualização é capaz de ajudar em cenários de rotulação, como é o caso de aprendizado ativo.

PUBLICATIONS

This thesis is the result of the following publications:

- F. C. M. Rodrigues, N. S. T. Hirata, A. A. Abello, L. T. de la Cruz, R. M. Lopes, and R. Hirata Jr. (2018) Evaluation of transfer learning scenarios in plankton image classification. *Proc.* 13^{th} *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, volume 5 (VISAPP), pages 359-366. SciTePress
- F. C. M. Rodrigues, R. Hirata, and A. C. Telea (2018) Image-based visualization of classifier decision boundaries. *Proc.* 31^{st} *Conference on Graphics, Patterns, and Images (SIBGRAPI)*, pages 353-360. IEEE
- M. Espadoto, F. C. M. Rodrigues, and A. Telea (2019) Visual analytics of multidimensional projections for constructing classifier decision boundary maps. *Proc.* 14^{th} *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, volume 3 (IVAPP), pages 136-145. SciTePress
- M. Espadoto, F. C. M. Rodrigues, N. S. T. Hirata, R. Hirata Jr., and A. C. Telea (2019) Deep Learning Inverse Multidimensional Projections. *Proc. EuroVis Workshop on Visual Analytics (EuroVA).* The Eurographics Association
- F. C. M. Rodrigues, M. Espadoto, R. Hirata Jr, A. Telea (2019) Constructing and Visualizing High-Quality Classifier Decision Boundary Maps. *Information* 10(9), pages 280-297. MDPI

CONTENTS

1 INTRODUCTION 1
  1.1 Classifier design in machine learning 1
  1.2 Decision Zones and Decision Boundaries 2
  1.3 Visualizing Decision Boundaries 3
  1.4 Research Questions 7
  1.5 Thesis Structure 9

2 RELATED WORK 11
  2.1 Machine Learning 11
    2.1.1 Logistic Regression 15
    2.1.2 Support Vector Machines 17
    2.1.3 k-Nearest Neighbors 20
    2.1.4 Random Forests 23
    2.1.5 Neural Networks 26
  2.2 Visual Analytics for Machine Learning 30
    2.2.1 High-Dimensional Data Visualization 31
    2.2.2 Dimensionality Reduction 34
      2.2.2.1 LAMP: Local Affine Multidimensional Projection 37
      2.2.2.2 t-SNE: t-Distributed Stochastic Neighbor Embedding 39
      2.2.2.3 UMAP: Uniform Manifold Approximation and Projection 40
    2.2.3 Inverse Projection Techniques 42
      2.2.3.1 Inverse LAMP 43
      2.2.3.2 RBF based Inverse 44
    2.2.4 Visual analytics techniques for classifier engineering 45
      2.2.4.1 Class-centric techniques 46
      2.2.4.2 Observation-centric techniques 46
      2.2.4.3 Architecture-centric techniques 48
    2.2.5 Conclusions 49

3 DEEP FEATURE EXTRACTION EVALUATION 51
  3.1 Introduction 51
  3.2 Problem Context 51
  3.3 Deep Feature Extraction 52
  3.4 Experiment Setup 53
  3.5 Datasets and Networks 55

3.5.0.1 Kaggle's National Data Science Bowl 56
3.5.0.2 ImageNet 57
3.5.1 CNN MODELS 58
3.5.1.1 AlexNet 58
3.5.1.2 Deep Sea's Model 59
3.5.2 Feature extraction 59
3.5.2.1 Deep features 59
3.5.2.2 Shape Features 60
3.6 Classifier Evaluation 60
3.7 Discussion and conclusion 63

4 CONSTRUCTING DECISION BOUNDARY MAPS 67
4.1 Dense maps 68
4.2 Decision Boundary Map Construction 71
4.2.1 Parameter setting 73
4.2.2 Implementation details 75
4.3 Experimental results 75
4.3.1 Segmentation dataset 75
4.3.2 MNIST dataset 77
4.4 Discussion 80
4.5 Conclusion 82

5 EVALUATING DECISION BOUNDARY MAPS 83
5.1 Preliminaries 84
5.2 Experiment Setup 85
5.3 Analysis of Evaluation Results 87
5.3.1 Phase 1: Picking the Best Projections 88
5.3.2 Phase 2: Refined Insights on Complex Data 91
5.4 Discussion 93

6 INVERSE PROJECTIONS FOR DECISION BOUNDARY MAPS 97
6.1 Inverse Projection by Neural Networks 98
6.2 Experiments and Results 99
6.2.1 Scalability in training and inference 100
6.2.2 Quantitative Assessment of Quality 101
6.2.3 Qualitative Assessment of Quality 102
6.3 Discussion and Conclusion 104

7 VISUAL REFINEMENTS OF DECISION BOUNDARY MAPS 107
7.1 Projection Filtering 108
7.2 Distance-enriched Dense Maps 111
7.2.1 Image-based Distance Estimation 112

7.2.2 Nearest-neighbor Based Distance Estimation 113
7.2.3 Adversarial Based Distance Estimation 114
7.2.4 Visualizing Boundary Proximities 115
7.2.4.1 Enridged Distance Maps 117
7.3 Discussion 118

8 END TO END EVALUATION 123
8.1 Semi Supervised Learning 123
8.2 Visual analytics for semi supervised learning 125
8.3 Manual labeling experiments 128
8.3.1 Classifiers description 128
8.3.2 Datasets description 129
8.3.3 Experimental set-up 131
8.4 Manual labeling results 132
8.4.1 Comparison with automatic labeling 134
8.5 Discussion 138

9 CONCLUSION 141
9.1 Deep Feature Extraction Evaluation 141
9.2 Decision boundary maps 142
9.3 Impact of direct projections on DBMs 142
9.4 Impact of inverse projection on DBM formation 143
9.5 Visual refinements of DBMs 143
9.6 End to end application 144
9.7 Future work 144

BIBLIOGRAPHY 147

CURRICULUM VITAE 161

ACKNOWLEDGMENTS 163

1

INTRODUCTION

1.1 CLASSIFIER DESIGN IN MACHINE LEARNING

Machine Learning (ML) is a branch of the Artificial Intelligence field whose main objective is to create algorithms that induce *models*, *i.e.*, *learn*, from data. ML methods are showing huge success when solving problems in many application areas such as medical diagnosis [62, 72, 109], recommendation systems [103], text classification [128], games [133], facial recognition [94], among many others. Recently, approaches known as *deep learning* achieved breakthrough results when solving hard computer vision problems [50], in special image classification [73].

Traditionally, machine learning tasks are divided into three major types [2]: supervised learning, reinforcement learning and unsupervised learning. *Supervised learning* uses *labeled* data, that is, data samples which have additional information associated to them. A successful algorithm in supervised learning must learn how to imitate the labeling process. In general, this process is done by presenting a set of examples (multidimensional vectors) and the expected output (label), the so-called *training set*, to the algorithm. Thus, a supervised learning method seeks for the model that optimally *fits* this mapping of points to labels. In *unsupervised learning*, the job of the algorithm is to induce a model from interactions with an environment. The idea is that good decisions, that is a series of actions that lead the system to a desirable state, are rewarded, while bad decisions are punished. Finally, *unsupervised learning* uses *unlabeled* data as input. In this case, the methods seek to find so-called structural characteristics of the data, such as clusters or its generative distribution.

Supervised learning can also be further split into two types of problems: classification and regression. In *classification* problems, the labels are discrete; labels usually have categorical values; and the set of possible labels is small. In regression problems, labels are in general continuous quantities. Another possible classification of ML techniques separates them into *active* versus *passive* learning. While a passive learner builds its knowledge from the information given to it, an active learner may query the user at training time, asking, for example, for the user to label certain dubious data samples [129].

This thesis mainly focuses on supervised learning, specifically on classification problems. In this context, the training phase of a ML algorithm can be usually reduced to the optimization task of finding a function g in a set of *hypothesis* $\mathcal{H}$ that minimizes a certain error measure for the training set. A learning algorithm is successful when the model obtained from it can be used to make inferences on new data, not originally part of the training set. To estimate how a predictor will behave in practice, *i.e.* when applied to solve a real world problem, it is a common practice to evaluate error on a so-called *test* or *validation* set, which is not used during the training.

1.2 DECISION ZONES AND DECISION BOUNDARIES

A classifier training process can be seen as dividing the high-dimensional data space into so-called *decision zones*. All samples from a given zone will be seen as similar, in the sense that they receive the same label. Such decision zones are separated by so-called *decision boundaries*. These are, in general, curved surfaces embedded in the high-dimensional space. Figure 1.1 illustrates the above concepts. Figure 1.1(a) presents a simple two-class 2D toy dataset composed of red squares and blue circles. Figure 1.1(b) shows a non-linear decision boundary (black curve) that partitions this space into two decision zones, corresponding to the red, respectively blue areas. The objective of a classifier algorithm is to find this border so that (a) all training samples, represented by the squares and circles Figure 1.1(a), fall within the correct decision zone; and (b) unseen (test, validation) samples, not available during the training phase, will also fall within their correct respective decision zones.

A successful training of a classifier can, thus, be seen as a process that leads to the construction of the "correct" decision boundaries (or decision zones, given that the former imply the latter and conversely). However, doing this is a fundamental open problem in machine learning for a number of reasons. Not exhaustively, these include

- a *fitting* problem: Given a labeled training set and a classifier model, how can one tune the internal parameters of the model so that decision zones emerge that *precisely* contain all samples of each class in the same decision zone? This is relatively easy to do when the number of classes is small, and when their training samples are well separated in the data space – that is, same-class samples are close to each other, while different-class samples are far away from each other. Simple classifiers such as logistic regression have limited freedom herein, since their decision boundaries are by construction hyperplanes. However, such classifiers are simple to train since they have few parameters. Slightly more complex classifiers, such as k-nearest neighbors (kNNs) are very similar with respect to their decision boundaries, as these correspond to the faces of a n-dimensional Voronoi diagram in the data space whose sites are the labeled samples. Conversely, more complex classifiers such as neural networks [76] have massive freedom in creating very complex decision boundaries, which can be curved piecewise-manifold surfaces embedded in the data space. However, such classifiers are more complex to train since they have thousands up to millions of parameters. In general, it is far from clear how to actually control the training process of a classifier so that it creates decision boundaries there where they are needed to obtain maximal accuracy on the training set;

- a *generalization* problem: Even for the favorable cases where one can train a classifier to optimally partition samples from the training set, it is far from clear whether the so-constructed decision surfaces will perform well on unseen test and/or validation data. When this is not the case, a process of fine-tuning of the classifier kicks in. Such a process involves many operations, such as tuning the training parameters and the so-called hyperparameters of the model itself; and changing the training set by adding, deleting, or changing samples and/or their labels. This tuning process can be tedious, lengthy, and is not guaranteed to converge to a good result.

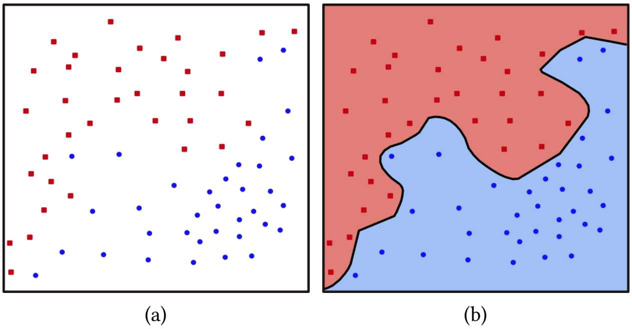


Figure 1.1: Simple toy dataset with two-class samples (a). A possible boundary decision a classifier might have induced from this training data (b).

1.3 VISUALIZING DECISION BOUNDARIES

As we have seen in the previous section, the (successful) training of a classifier is intimately related to the process of constructing the correct decision boundaries. However, as also explained there, the complexity of this process is intuitively proportional with the complexity of the decision boundaries. In the above, we can roughly partition classifiers (and the above-mentioned challenge) into two classes, as follows.

Explicit boundaries: These are classifiers whose models are simple enough so that one can compute, and reason about, their decision boundaries in an explicit way. For instance, linear classifiers induce from a n-dimensional dataset, a n-dimensional hyperplane that splits data space into regions. Given a data point in this space, the question of to which class it belongs could be answered simply by checking whether this point lies to one side or the other of the hyperplane. Moreover, the confidence of classification can be well approximated by the distance of the point to its closest hyperplane boundary, something which is simple to compute and reason about analytically. The same can be said about other simple classifiers such as kNNs, given that one can explicitly compute, and reason about, following their underlying Voronoi diagram model: Given a new data point, it will be assigned the same class as the point closest to it in the training set. However, the decision boundaries of such simple classifiers are also very limited in their flexibility to "squeeze between" complex distributions of labeled samples present in the data space so as to successfully partition them.

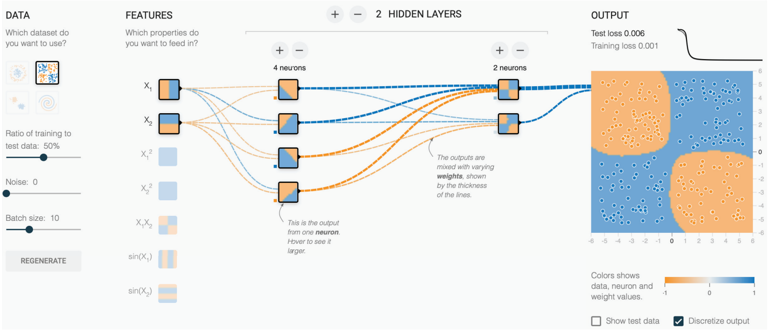
Implicit boundaries: Formally speaking, even complex classifiers such as neural networks do create explicit decision boundaries. The problem is that one cannot extract an analytical formulation of such boundaries, as that would involve complex equations having thousands up to millions of parameters (the network weights), which also do not have an intuitive meaning. Hence, the decision process used by the classifier is not simple to understand. For instance, it can be challenging for a user to understand why two apparently similar data points (*i.e.*, having similar coordinates in the data space) were assigned different labels – that is, why a decision boundary "squeezed through" these two points. Even though, formally, a decision boundary is explicitly computed by the classifier, in practice, one can only *implicitly* assess where this boundary is, by evaluating the classifier with samples (*e.g.*, test and validation) and seeing which labels are produced. Hence, we cannot (easily) reason analytically about the decision boundaries for such models in the same way we could for the explicit decision boundaries of simpler classifiers. In the field of machine learning, such models are commonly regarded as *black boxes*, given the difficulty to explain the logic behind some of their decisions in simple terms.

The black-box nature of deep learning models is highly related to how they are trained. Take image classification tasks as an example. While a "classic" image classification pipeline requires manually designed features to be defined, current deep learning approaches take images directly as input and output a class label. In this case, features are implicitly learned during training process and they may not match exactly domain experts definition [166]. In other words, classic methods would require a user to specify what patterns the classifier should seek for, *e.g.*, roundness, corners, or specific textures; while deep learning models are trained in a end-to-end fashion, defining as the process unfolds which features are important. Understanding decision boundaries is, for the deep learning case – or more generally, for cases where the input features are numerous and/or not easy to reason about – additionally challenging.

Given the above mentioned challenges of complex classifiers, which do not allow one to analytically study their decision boundaries, how can we approach this problem?

Visualization is a field of Computer Graphics that seeks to convey into images information about data. Visualization is mainly focused on creating insightful figures that help users understand and even discover new patterns and behavior present in a given dataset. More generally, visualization aims to leverage the user's visual system ability to recognize complex patterns to solve data-related problems that cannot be easily mapped to explicit queries or analyses suitable to computer automation.

Visualization is traditionally divided into two operational subfields: Scientific visualization (*SciVis*) focuses on addressing problems related to data which is spatially embedded in two or three dimensions and comes from continuous processes such as measurements of physical quantities. SciVis has shown huge success, being vastly applied on data from medical science, civil engineering, geosciences, and physics [54, 150]. Information Visualization (*InfoVis*) focuses on addressing problems related to data which is either embedded in high-dimensional spaces, or which does not even have a spatial nature [90]. Moreover, InfoVis data can be of more than the quantitative type covered by SciVis – its data values can be of any type, *e.g.*, ordinal, integral, categorical, text, images, video, or relations. Such datasets include social network feeds, and news (text data), hospital patient data, software quality metrics, and business data (tabular data), information flows in a network, flight destinations, subway maps, and software systems structure (relational data). In general, handling InfoVis data is considered harder than handling SciVis data, given its typical non-spatial nature, high dimensionality, and mix of attribute types



Ratio of training to test data: 50%

Noise: 0

Batch size: 10

Test loss 0.006

Training loss 0.001

This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

Colors shows data, neuron and weight values.

☐ Show test data ☑ Discretize output

Figure 1.2: Tensorflow visualization of decision boundaries for a simple neural network classifier for 2D two-class data. See Sec. 1.3.

Visual analytics (VA) builds up on techniques from SciVis and InfoVis by creating interactive methods and tools to allow users to explore a complex data space and underlying hypotheses (questions) [69, 152]. Key to VA is the *exploration* process: Simply put, while both SciVis and InfoVis offer tools and techniques where users can interactively change various parameters to create different views of the data, VA places this exploratory process, and its support by interactive tools and techniques, in the center. As such, VA solutions are, in general, specific to a given problem and application, aiming to provide both tools and *workflows*, *i.e.*, procedures that empower users to obtain the answers to their problems by methodologically following an iteration of hypothesis posing, (in)validation, and refinement steps.

Given the challenge of training classification models for cases where boundaries cannot be computed and/or reasoned explicitly about; and given the power of InfoVis for helping users to reason with complex, high-dimensional, and abstract data – characteristics that our ML classification data fully share – we advocate that it is interesting to consider InfoVis and VA for depicting, explaining, and understanding such decision boundaries.

The added value of visualization for this task is illustrated by Figure 1.2, generated with TensorFlow [138]. Here, a simple two-dimensional dataset is considered. In this dataset, points are of two classes, and the goal is to train (and understand) a neural network classifier for this data. The actual points in the training set follow a checkerboard pattern, as shown by the orange and blue points in the visualization at the right. A simple two-hidden-layer neural network is used to learn a classification model from this pattern. The image to the right shows, in the background, a coloring of the two-dimensional data space by the labels that the classifier would assign to every (x, y) possible data point in this space. As visible, this shows the apparition of two decision zones – a compact one for the blue label and one consisting of two blobs for the orange label, respectively. The two zones nicely contain the training samples of the two classes, *i.e.*, the classifier has maximal accuracy on the training set. Besides that, activation maps are displayed as thumbnails for all units in all layers, showing which class label these would produce for any possible data point. Additionally, users can vary the architecture, training and test data, and hyperparameters of the network interactively and see how the visualization changes.

Figure 1.2 is a good example of how one can use visualization of the decision boundaries and decision zones to understand the operation of a classifier. Unfortunately, the underlying techniques used to produce this visualization are not generalizable to data *higher than two dimensions*, nor is the visualization clearly scalable for datasets having thousands of samples or more. This example, and the previous considerations outlined in this section, leads us to the formulation of our main research questions described next.

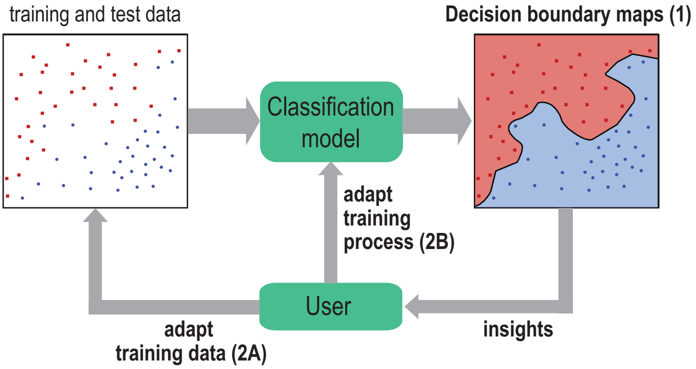


Figure 1.3: Workflow of this thesis capturing its two research questions – visualizing the decision boundaries (1) and creating a VA loop for gathering insight to improve training (2A,2B). See Sec. 1.4.

1.4 RESEARCH QUESTIONS

In this thesis, our focus is on developing new visualization and visual analytics techniques specifically suited for understanding and improving ML classifiers. Our first research question can be thus stated as:

How to use information visualization and visual analytics to get more insight into a classifier's operation and performance?

There are many ways in which the operation of a classifier can be thought of. Similarly, performance of a classifier can be measured by many different metrics [139] and subject to many parameters, *e.g.*, the size, quality, and distribution of samples in the training and test sets, and setting of the many hyperparameters of the underlying model. However, in our work, we focus on the link outlined in Sec. 1.3 between a classifier's operation and the decision boundaries it constructs. Specifically, we further refine the above question to become:

How can we depict the decision boundaries of a classifier and use these to understand its operation and performance?

Answering the above question helps the users of a classifier to understand why, for instance, a classifier performed well (or not) in a given situation, that is, a given pair of training and test sets and a given setting of its hyperparameters. Information visualization is the key enabling instrument for answering this question, as we will need to create explicit, visual, depictions of the high-dimensional and complex decision boundaries, decision zones, and related quantities, *e.g.* the distance of samples to their closest decision boundaries. Simply put, answering this question aims to create, for any dataset and classification problem, images as simple as the ones shown in Fig. 1.1b. Figure 1.3(1) outlines this research question: We start with training and test data, and create a classification model. Next, we visualize its decision boundaries to get more insights in its operation.

However, in the case classifier performance is not optimal, the natural goal is to next try to improve this situation. Hence, we state our second research question as follows:

How to use visual analytics, supported by decision boundary visualizations, to improve a classifer?

Visual analytics will also play an important role in answering this question (see also Fig. 1.3(2)): Based on the visual representation proposed by the answer to the first question (step (1) in the figure), we will add interactive mechanisms to inspect classification problems, suggest their causes (hypotheses), and allow the user to iteratively explore several alternative solutions to these problems, such as changing the training data (step (2A) in the figure) and/or changing the model's hyperparameters (step (2B) in the figure), so as to ultimately leading to an improved classifier.

We aim to answer our research questions in the broadest possible sense. This adds several so-called non-functional requirements (to use a well-known terminology in requirements engineering) to the functional ones related to depicting decision boundaries and using this information to understand and improve a classifier. These non-functional requirements are as follows:

- *Generality:* Our solutions should be applicable to the widest possible family of classifier techniques.
- *Genericity:* Refining generality, we ideally want to treat a classifier as a functional "black box" that receives training and test data and outputs a sample labeling. That is, our solutions should not need to know the internals and/or implementation details of a specific classifier technique. This way, we can apply them easily to satisfy the generality requirement.
- *Ease of use:* Our solutions should be easily deployable and usable by users who have only limited knowledge of machine learning, no specific knowledge of the internals of a particular classifier technique, and no intimate knowledge of information visualization technicalities.
- *Scalability:* Our solutions should scale well, both computationally and in terms of the screen size required to display the results, to datasets containing tens of thousands of samples (or more), each having hundreds up to thousands of dimensions.

- *Replicability:* Our results should be easily replicable by interested users. This imposes subsequent constraints on their ease of use, description of their parameter setting, and availability of third-party components of our end-to-end pipeline, such as classification technique implementations.

As we shall see starting with Chapter 4, answering our research questions under these assumptions will introduce additional technical and conceptual challenges that need specific algorithms and techniques to be developed to be addressed.

1.5 THESIS STRUCTURE

Following the two research questions outlined in Sec. 1.4, we structure the remainder of this thesis as follows:

Chapter 2 presents the necessary theoretical background on machine learning, information visualization, and visual analytics that are relevant to our work. This chapter also discuss existing visualization techniques that have machine learning in their focus, outlining limitations thereof that we will aim to alleviate.

Chapter 3 presents a typical ML scenario in which one wants to develop a good classifier for a specific dataset, starting from a pre-trained model on different data. The main contribution of this chapter is the comparison of how different Deep Neural Networks used as feature extractor impact classification accuracy. The material in this chapter also serves as a concrete illustration for the types of difficulties that machine learning practitioners face when trying to understand, and further tune, "black box" classifiers, and hence supports the claims for the added value of exploring the visual analysis of decision boundaries which are expanded on further in the thesis.

Chapter 4 presents the idea of computing and depicting the inferred decision boundaries of any given classifier by a *dense*, or image-based, approach. Two possible directions to achieve this objective are discussed, and one of them is chosen upon reflections on strengths and weaknesses of both with respect to the non-functional requirements outlined at the end of Sec. 1.4. The output of this technique is an image called a Decision Boundary Map (DBM), that explicitly depicts the continuous nature of the actual decision zones and their decision boundaries that a classifier implicitly constructs during its training.

Chapter 5 explores in detail the DBM concept and idea introduced in Chapter 4. Specifically, since the computation of DBMs depends crucially on the choice and parameterization of so-called dimensionality reduction techniques, or projections, we explore here empirically which such techniques provide the best results for DBM computation for a range of datasets and classifier models. The chapter concludes proposing good configurations for the DBM computation process which we will use next in our work.

Chapter 6 explores a second (and last) technical aspect of the DBM computation, namely the use of inverse projection methods. Together with the (direct) projection methods, whose effect is discussed in Chapter 5, inverse projections equally affect the quality and computational effort required to create DBMs, thus directly relate to our non-functional requirements. We explore here the suitability of several inverse projection methods known in the literature for DBM computation and outline their limitations. Considering these, we also propose a novel method for computing inverse projections based on deep learning which has favorable quality and speed properties as compared to existing methods, and can be also used in any generic inverse projection task. We compare our method with existing ones for the task of computing DBMs.

Chapter 7 refines the basic DBM model introduced in Chapter 4 (and further refined in Chapters 5 and 6). While the previous two chapters focused on computational aspects of DBMs, we now focus on their visual presentation. Specifically, we propose a set of mechanisms to increase the amount of information displayed by DBMs by highlighting regions and distances on these maps that serve various tasks related to understanding the behavior of a classifier model. Hereby, we conclude our addressing of the first research question, which covers Chapters 4 to 7.

Chapter 8 addresses the second research question by proposing a visual analytics tool that uses the DBMs created with techniques presented in the previous chapters to support the process of improving a classifier. Our tool and underlying methodology allow detecting classification problems on the DBM, determining the order in which the user may want to address these problems, and subsequently perform user-driven labeling to create a better training set. We demonstrate the working and added value of our VA approach on several classification problems involving real-world data.

Finally, Chapter 9 presents a brief summary of our contributions and how they are related to each other to attempt answering the two research questions proposed above. We close this chapter, and the thesis, by proposing directions for future work.

2

RELATED WORK

In this chapter we present a summary of the theoretical background required for this thesis, as well as related works which aim at a similar objective to ours, that is, the use of visualization to support the understanding (and improvement) of machine learning classifiers as outlined in Chapter 1. In Section 2.1 we give the basic definitions related to Machine Learning that we are going to use throughout this thesis. Besides that, a brief explanation of the main classification techniques that we use in the remaining of this thesis is presented. Next, in Section 2.2 we discuss information visualization techniques for machine learning. In particular, we dedicate attention to Dimensionality Reduction (DR) and Inverse Projection (IP) techniques, which are fundamental to the interpretability of a model. We dedicate special attention to several such techniques, such as t-SNE and UMAP, as these are the ones most frequently used in the other chapters of the thesis. Similarly, we explain in detail two inverse projection methods, as they will be referred to in Chapter 6 when we compare our novel inverse projection method to them.

2.1 MACHINE LEARNING

This brief summary of supervised machine learning is not intended as a complete reference, instead its purpose is to contextualize different techniques that will be needed in the next chapters. For a complete background on machine learning, we refer to standard textbooks in the area [2, 16, 129].

As mentioned in Chapter 1, classifiers are part of a branch of supervised learning that attempts to induce a mapping from data points to labels.

Preliminaries: Formally put, let $\mathcal{D}$ a dataset of N ordered pairs $(\mathbf{x}_i, y_i)$, *i.e.* $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, generated by a real or simulated phenomenon modeled as a distribution $P(y|\mathbf{x})$. The set of observations, or samples, is denoted as $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, where $\mathbf{x}_i \in \mathbb{R}^n$, and n can be tens, up to thousands, of dimensions. Each such observation $\mathbf{x}_i = (x_i^1, \ldots, x_i^n)$ can be thus seen as a n-dimensional feature vector. We call x_i^j, with $1 \leq j \leq n$ the j^{th} attribute, or dimension, or variable, of point $\mathbf{x}_i$. The sets $\mathbf{X}^j = \{x_i^j\}$, $1 \leq i \leq N$, are called the *attributes*, or dimensions, of the entire set of samples $\mathcal{X}$. Hence, $\mathcal{X}$ can be represented as a data table with rows corresponding to observations $\mathbf{x}_i$ and columns corresponding to the attributes $\mathbf{X}^j$, respectively. For each observation $\mathbf{x}_i$, a label y_i can be attributed to it accordingly to distribution $P(y|\mathbf{x})$. Let $\mathcal{Y} = \{y_1, \ldots, y_N\}$ be the set of samples' labels.

As outlined in Chapter 1, for classification problems, these samples take values typically in a categorical set whose values indicate the different classes (types) of samples. The goal of supervised learning is to find a function $g : \mathcal{X} \rightarrow \mathcal{Y}$, that best approximates the outcomes of $P(y|\mathbf{x})$ from the finite dataset $\mathcal{D}$. The search for g is performed by a learning algorithm on the space of the Hypothesis Set $\mathcal{H}$, in general guided by an *error measure* computed on the candidate functions $h \in \mathcal{H}$.

Error measures: An error measure gauges how well g represents the outcomes of $P(y|\mathbf{x})$. An error function can be defined for the whole dataset, and denoted hence as $E(g(\mathbf{X}), \mathcal{Y})$, or alternatively in a point-wise fashion, denoted as a function $e(g(\mathbf{x}_i), y_i)$. Supervised machine learning algorithms will seek for a function that *minimizes* the error, often also called *loss* or *cost*. Besides guiding the learning algorithm on the exploration of $\mathcal{H}$, error measures also serve the purpose of evaluating model's performance.

Performance measuring: For classifiers, a measure of performance that is similar to computing the error over a finite dataset is *accuracy*. Accuracy of a classifier is simply the count of correct guesses over the total number of samples in the dataset, *i.e.* $Acc = \frac{1}{N}\sum_{i=1}^{N} [\![g(\mathbf{x}_i) = y_i]\!]$, where $[\![\cdot]\!]$ is one if $g(\mathbf{x}_i) = y_i$ and zero otherwise. While accuracy is simply the percentage of correct guesses, error functions in general depend of the computation of probabilities and may not be as easy to grasp. As such, accuracy values are simpler to interpret, and used more often to understand, classifiers than the respective error values.

Although an error function computed on $\mathcal{D}$ guides the search for the best model, a machine learning program is useful if it is capable of making correct inferences when applied in a production environment, specially on new, *unseen*, data. In other words, in machine learning we are interested in finding a model that will *generalize* to samples outside the training set. A simple algorithm that memorizes every entry in $\mathcal{D}$ would not be interesting even with $E = 0.0$, as the out of sample error will most likely be high, *i.e.* this model would not generalize. In practice, error measured on the set of samples using for training is not completely useless, as the memorization algorithm just mentioned is not practical, and is still used to estimate a model's performance.

Training, testing, and validation: When the performance measured in production of a trained model does not reflect the error obtained during training, we say *overfitting* has occurred. That is, the model works very well on the training data, but generalizes poorly on different data. A common approach in machine learning to avoid overfitting is to split

$\mathcal{D}$ into two sets, one for training and another for testing. As the samples in the test set were not seen by the model during the training phase, computing the error or accuracy on this set serves as a better proxy to the true classifier error expected in production mode.

A slight variation of the above is to split $\mathcal{D}$ into three sets: training, validation and test. In this setting, a model is trained using the samples from training, but different *hyperparameters* or configurations can be experimented with, reporting the achieved accuracies or error values on the validation set. This dataset partition can be used even to compare and choose between different models. When the best configuration is chosen, the final model is trained again using all the samples from both training and validation sets and final accuracy or error is then reported on the samples from the test set, which were not used before.

A generalization of this technique is called *cross validation*. Cross validation consists in partitioning a dataset $\mathcal{D}$ of size N into K sets of sizes $\frac{N}{K}$, which is known as K-fold cross validation. In this setting, there are K configurations in which one of those sets is left out for validation, and the remaining $K - 1$ sets are used to train a model. The set left out is used to evaluate performance (accuracy or error) for each configuration, and since there are K of them, a good out of sample performance can be estimated averaging their values. In one extreme scenario with $K = N$, sets of one element are set up, leading to a good out of sample accuracy estimate. However, this *leave one out* strategy is impractical due to the computational resources needed, thus in general, $3 \leq K \geq 10$ is a commonly chosen value. Such cross validation strategies seek to balance the search performed by the learning algorithm to minimize error in the data by estimating the true expected error the model will present when applied in real scenarios. Hence, cross validation techniques aim at improving generalization.

Regularization: Besides cross validation, another heuristic commonly employed to prevent overfitting is *regularization*. Regularization consists in limiting the search for candidate functions in the hypothesis set $\mathcal{H}$, allowing to train on a space of complex, *i.e.* flexible, functions, but avoiding candidates that would overfit to the training data. Arguably, the most usual path to implement a regularization mechanism into a learning algorithm is by adding a penalty term to the error function in order to prevent undesirable configurations, as $E_{new}(g(\mathcal{X}, \mathcal{Y}) = E_{old}(g(\mathcal{X}), \mathcal{Y}) + \lambda R(g)$, where $R(g)$ is the regularization term and λ a weight to gauge the importance of regularization to the task. An usual example of such a restriction during the training phase of a parametric model is enforcing the preference for simpler candidate functions by penalizing large parameter values. In this type of scenario, that is, inducing a parametric model $g(\mathbf{x})$ that depends on parameters $\mathbf{w}$, a commonly applied regularization is $R(g) = \mathbf{w}^T\mathbf{w}$. By penalizing large values of $\mathbf{w}$, simpler models are expected to be inferred as some elements of the vector of parameters will be too small or even zero. Intuitively, regularization can be seen as adding a certain "stiffness" to the function learned by the model, so as to balance between fitting well the training data, but allowing sufficient flexibility to approximate well the test (unseen) data.

Two other types of regularization mechanisms that are commonly used in ML, in particular when training neural networks, are *early stopping* and *dropout* [141]. Early stopping can be used for iterative methods and consists in monitoring the error on a validation set at each iteration, and halting the training if this error starts to increase. Highly adaptable and flexible models could still decrease the error on the training set, but if the validation error increases, then the model must be overfitting to the training data. The second heuristic mentioned, dropout, consists in removing each neural network's node, with a probability $1 - p$, at each training iteration. In this setting, the parameter associated with that node will not be updated in this training phase, but the node will be reinserted into the network in the next iteration. During testing, or deployment, each node output will be weighted by p, the probability that the node would not be removed in any given iteration.

Choice of techniques: At this point, we revisit our main research question introduced in Sec. 1.4. As stated there, we aim to use visualization to get insight into a classifier's operation and performance. Also, we aim to do this in a general and generic fashion, *i.e.*, without having to rely upon implementation details or knowledge of the operation of specific classification techniques. However, during this process, we will also need to *test* our proposed solutions, and for this purpose we need to choose a number of classification techniques. While, in theory, any subset of the universe of all possible classification techniques would do for testing, we prefer a reasoned selection in order to better assist our testing task. Specifically, we will choose classifier techniques to test our visualization solutions against based on the following criteria:

- *Relevance:* We aim to include techniques which, albeit not among the modern state-of-the-art, are well battle-tested, deployed, and known in the ML community. Simple techniques will also help us in *understanding* how our visualizations works. Indeed, since for such techniques, we do have a good understanding of the decision boundaries they create, as outlined in Sec. 1.3, we can use this "ground truth" knowledge as a way to gauge our visualization results.

- *Variation:* We aim to include techniques that use very different underlying implementations, rather than using many instances of the same type of technique (*e.g.*, different architectures of a deep learning model). This way, we arguably "sample" the universe of classification techniques better, and thus test our visualization proposals more exhaustively.

- *Replicability:* We aim our work to be readily replicable, both in terms of reproducing the results of our experiments, but also allowing interested users to set up similar pipelines using the same software implementations of *e.g.* classification techniques (see the replicability requirement in Sec. 1.4). Hence, we favor classification techniques that exist as part of well-known, well-documented, publicly available, ML libraries, such as *scikit-learn* [104] and Keras [23].

Based on the above requirements and rationale, we have selected several classification techniques to investigate next in our work. These are described in detail in the following. We order these in increasing order of their complexity – that is, start by the arguably simplest techniques and end by the more complex, but also more powerful, ones.

2.1.1 *Logistic Regression*

Linear models are functions of the form $l(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. In this section, we assume that $\mathbf{x}$ is a bias-augmented vector, that is, $\mathbf{x} = (1, \mathbf{x}')^T$, where $\mathbf{x}'$ is original feature vector in the dataset and the first coordinate of $\mathbf{w}$ is the bias, or independent term. Where $\mathbf{w}$ is a vector of real-valued parameters induced by the learning algorithm. Such functions define a *hyperplane* in which every point $\mathbf{x}$ that satisfies $l(\mathbf{x}) = 0$ lies on it. In addition, one can check whether $\mathbf{x}$ lies to one side or another of this hyperplane by evaluating the signal of $l(\mathbf{x})$. In this case, the simplest classification rule possible for a two-class problem with labels -1 and $+1$, would be given by the function

$$g(\mathbf{x}) = \begin{cases} -1, & \text{if } l(\mathbf{x}) < 0; \\ +1, & \text{otherwise.} \end{cases} \tag{2.1}$$

Such classification rule imposes a hard threshold on the linear combination of model's parameters and input values. A smoother version, more used in practice, outputs class probability values in the range $[0, 1]$. Intuitively put, such probabilities describe how certain the model is that a given sample is of a given class. *Logistic Regression* is likely the best known probabilistic model of this type. For this type of learning method, the hypothesis set $\mathcal{H}$ is composed of functions of the form $\theta(\mathbf{w}^T\mathbf{x})$, where $\theta : \mathbb{R} \rightarrow [0, 1]$ is defined as $\theta(s) = \frac{e^s}{e^s+1}$ is a logistic function and e is Euler's number. As stated earlier in Sec. 2.1, the objective of supervised machine learning is to approximate $P(y|\mathbf{x})$. For that, we can use the function $h(\mathbf{x})$ just described, leading to the probability

$$P(y|\mathbf{x}) = \begin{cases} \theta(\mathbf{w}^T\mathbf{x}) & \text{for } y = 1; \\ 1 - \theta(\mathbf{w}^T\mathbf{x}) & \text{for } y = -1. \end{cases} \tag{2.2}$$

Given that we assume class labels to be either +1 or −1 and noting that $1 - \theta(x) = \theta(-x)$, Eqn. 2.2 can be compactly written as

$$P(y|\mathbf{x}) = \theta(y\mathbf{w}^T\mathbf{x}).$$

Hence, the probability of correctly assigning labels to every sample in a finite dataset is given by

$$\prod_{n=1}^{N} \theta(y\mathbf{w}^T\mathbf{x}). \tag{2.3}$$

Thus, to fit the best model according to the logistic regression rules just presented, it is necessary to find the set of parameters $\mathbf{w}$ that maximize the product in Eqn. 2.3. This is equivalent to minimizing the following expression:

$$\frac{1}{N}\sum_{n=1}^{N} ln\left(\frac{1}{\theta(y_n\mathbf{w}^T\mathbf{x}_n)}\right). \tag{2.4}$$

Replacing the logistic function defined earlier in Eqn. 2.4, an error function for a Logistic Regression learning model can be defined as

$$E(w) = \frac{1}{N}\sum_{n=1}^{N} ln\left(1 + e^{-y_n\mathbf{w}^T\mathbf{x}_n}\right). \tag{2.5}$$

The parameters $\mathbf{w}$ that minimize the error function in Eqn. 2.5 can be found with ease applying standard optimization methods. The usual choice in this context is to apply *gradient based* methods, such as Gradient Descent, or Stochastic Gradient Descent. As the gradient of a function is a vector that points to the direction of greatest increase, those methods consist in iteratively updating the set of parameters towards the *opposite* direction of the gradient of the error function, effectively minimizing the error. Such methods are capable of returning *global* minimum of a convex function, as is the case for LR error function. When the function is not convex, the absolute minimum value is not guaranteed to be found, *i.e.*, a *local* minimum point is found (we return to that on Sec. 2.1.5, when discussing Neural Networks). We refer to [18] for a more extensive explanation of gradient methods, in special Stochastic Gradient Descent, in ML.

Arguably, Logistic Regression's key added values are its simplicity of implementation and ease of understanding how it operates. Basically, for two-class problems, we can think of it as drawing a hyperplane that best separates samples of the two classes, according to a probabilistic definition of separation. Using this intuition, we can also immediately see the limitations of this technique – namely, the fact that it cannot separate (well) more complex sample distributions. Nevertheless, the method's simplicity and intuitiveness make it an ideal candidate for testing our visualization methods introduced later in this thesis.

2.1.2 *Support Vector Machines*

Support Vector Machine (SVM) is a popular linear classifier frequently used in practice. While a Logistic Regression model seeks for parameters that maximize a probability measure, SVM employs a geometric approach and seeks for the parameters that lead to the hyperplane that best separates data. In this sense, the best separating hyperplane is the one that *maximizes* the distance it is placed to the data points. Intuitively, the larger the distance from the separator plane to the nearest data points the more robust the model is to noise and errors in general.

We use next a slightly different notation from Sec. 2.1.1 in order to simplify the presentation, making the *bias* term b of the parameters explicit. We define a hyperplane as $h = (b, \mathbf{w})$, where $\mathbf{w}$ is a normal vector to the hyperplane, and any point $\mathbf{x}'$ that lies on the plane must satisfy $\mathbf{w}^T\mathbf{x}' + b = 0$. The distance between any point $\mathbf{x}$ and the hyperplane h can be computed as the scalar projection of the vector (proj) $\mathbf{x} - \mathbf{x}'$, where $\mathbf{x}'$ is any point that lies on the surface of the hyperplane. Thus, this distance can be computed as

$$\text{dist}(\mathbf{x}, h) = \text{proj}(\mathbf{x} - \mathbf{x}')\frac{\mathbf{w}}{\|\mathbf{w}\|} = \frac{\left|\mathbf{w}^T(\mathbf{x} - \mathbf{x}')\right|}{\|\mathbf{w}\|} = \frac{\left|\mathbf{w}^T\mathbf{x} + b\right|}{\|\mathbf{w}\|}. \tag{2.6}$$

Assuming −1 and +1 as class labels for a two-class problem, and using Eqn. 2.6, we can write the distance from any data point $\mathbf{x}_i$ to the hyperplane h as

$$\text{dist}(\mathbf{x}_i, h) = \frac{y_i(\mathbf{w}^T\mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

A hyperplane that completely separates a dataset into two sets of points having different labels is such that $y_i(\mathbf{w}^T\mathbf{x}_i + b) > 0$ for every pair $(\mathbf{x}_i, y_i) \in \mathcal{D}$. This inequality can be modified by normalizing the hyperplane's parameters ($\mathbf{w}$ and b). In particular, one can normalize a hyperplane to ensure that $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$. This can be accomplished by modifying the weights as follows: Let $v > 0$ be the smallest value of $\mathbf{w}^T\mathbf{x}_i + b$ among all data points, create another hyperplane

$h' = (b/v, \mathbf{w}/v)$. In this case h and h' are equivalent regarding classification rules, *i.e.*, checking whether samples fall on one side or the other of the hyperplane, but now the smallest value of the signal $\mathbf{w}^T\mathbf{x}_i + b$ is 1.

Considering such normalized hyperplane, the distance of the closest data point to it is given by

$$\text{dist}(\mathbf{x}, h) = \frac{1}{\|\mathbf{w}\|}. \tag{2.7}$$

Thus, finding the hyperplane that separates the dataset and such that it is of maximum distance to the closest point is equivalent to maximizing the quantity in Eqn. 2.7. The maximum-margin separating hyperplane can be similarly found solving the minimization problem

$$\begin{aligned} \underset{b,\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{subject to} \quad & y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1. \end{aligned} \tag{2.8}$$

The formulation in Eqn. 2.8 assumes that data is linearly separable. For non-separable datasets, the problem above can be reformulated by introducing so-called slack variables $\xi_i \geq 0$ that allow a number of data points to violate the separability condition $y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i$. This way, the minimization problem can be written as

$$\begin{aligned} \underset{b,\mathbf{w},\xi}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n \\ \text{subject to} \quad & y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 - \xi_n \\ & \xi_n \geq 0 \end{aligned} \tag{2.9}$$

In Eqn. 2.9, C is a user-defined parameter that controls the amount of separability violation that is acceptable.

Optimization problems may be viewed from two closely related perspectives: a primal problem and a dual problem. In general, the solution for the dual problem provides a lower bound to the primal. The solution for the primal optimization problem posed in Eqn. 2.8 and its dual version are equivalent, that is, the hyperplane obtained from one of them is exactly the same that would be obtained from the other. Consider de canonical quadratic minization problem below:

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{u}^T Q\mathbf{u} + \mathbf{p}^T\mathbf{u} \\ \text{subject to} \quad & \mathbf{a}^T\mathbf{u} \geq c, \end{aligned}$$

which is equivalent to the following:

$$\underset{\mathbf{u}}{\text{minimize}} \quad \frac{1}{2}\mathbf{u}^T Q\mathbf{u} + \mathbf{p}^T\mathbf{u} + \max_{\alpha \geq 0} \alpha(c - \mathbf{a}^T\mathbf{u}),$$

where α is the Lagrangian multiplier. The problem above can be compactly written as:

$$\min_{\mathbf{u}} \max_{\alpha \geq 0} \mathcal{L}(\mathbf{u}, \alpha), \tag{2.10}$$

where $\mathcal{L}(\mathbf{u}, \alpha) = \frac{1}{2}\mathbf{u}^T Q\mathbf{u} + \mathbf{p}^T\mathbf{u} + \alpha(c - \mathbf{a}^T\mathbf{u})$ is the Lagrangian function.

Hence, the Lagrangian dual formulation for the linearly separable SVM presented in Eqn. 2.8 is:

$$\mathcal{L}(b, \mathbf{w}, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{n=1}^{n=N} \alpha_n(1 - y_n(\mathbf{w}^T\mathbf{x}_n + b)).$$

From Eqn. 2.10, we have that we must minimize $\mathcal{L}$ with respect to b and $\mathbf{w}$ and maximize it with respect to α. By deriving and setting the derivatives to zero, we obtain the following Lagrangian:

$$\max_{\alpha \geq 0} \mathcal{L}(\alpha) = -\frac{1}{2}\sum_{m=1}^{N}\sum_{n=1}^{N} y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T\mathbf{x}_m + \sum_{n=1}^{N}\alpha_n. \tag{2.11}$$

Solving this maximization problem will return the optimal parameters α, which can be used to obtain the hyperplane parameters $\mathbf{w}$ and b.

The dual formulation for SVM allows non-linear transformations to be easily perfomed, thus resulting in a non-linear classification rule, through the technique known as *the kernel trick*. Notice that Eqn. 2.11 depends on the dot product $\mathbf{x}_n^T\mathbf{x}_m$ and hence one can use *kernel* functions to obtain non-linear separating hyperplanes. A kernel function has the following form

$$K_\phi(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T\phi(\mathbf{x}'),$$

where ϕ is a non-linear transformation function. The main benefit of using the kernel trick is that the actual transformation does not need to be computed. For instance, consider the following non-linear transformation

$$\phi(x) = \exp(-x^2)\left(1, \sqrt{\frac{2^1}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \sqrt{\frac{2^3}{3!}}x^3 \ldots\right)$$

it is not feasible to be computed in practice, as it is an infinite dimensional transformation. However, the inner product between two vectors that would be transformed to this space can be obtained efficiently as

$$K_{\phi}(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \left\|\mathbf{x} - \mathbf{x\prime}\right\|^2\right).$$

Hence, the dual formulation of SVM enables for the application of the kernel trick, allowing non-linear data classification in a computationally efficient manner.

Overall, Support Vector Machines can be seen as generalizing the hyperplane separation idea of Logistic Regression models by using a different definition of what a “good” separation hyperplane is, based on the minimal separation distance. While a discussion of the exact differences is not within our scope, one intuitive way to summarize these is to note that Logistic Regression uses *all* the samples equally to determine the placement of the hyperplane. In contrast, for Suppot Vector Machines the only data points that affect how the method induces the hyperplanes are the ones closest to it, *i.e.* the so-called support vectors.

In our context, we consider Support Vector Machines as a good candidate for testing our visualization proposals for the same reasons we included Logistic Regression – namely, intuitive understanding of the algorithm, presence in the ML literature and practice, and readily available implementations.

2.1.3 *k-Nearest Neighbors*

Different from the two previous classifiers presented, *k-Nearest Neighbors* (kNN) is a *nonparametric* learning model. While Logistic Regression and SVM go through an optimization process to fit a set of parameters that are combined with data points to form a classification rule, thus are regarded as parametric models, kNN does not depend on such parameters to be used on its classification function. Instead, the classification function depends on distances computed in data space, returning the label common to the majority of its k closest samples in the dataset. For $k = 1$, such a classification rule induce a known tessellation of data space known as *Voronoi diagrams* [15]. Simply put, the $n$dimensional space in which the samples (called sites in Voronoi terminology) exist is split into convex polyhedra, called cells. Each cell contains a single sample. Cell faces are hyperplane segments (polygons) that are at equal distance between two samples. That is, all points within a cell are closest to that cell’s site than to any other sites in the dataset.

Figure 2.1 illustrates Voronoi diagrams. The first image (a) shows a 2D site-set (white points) and their corresponding cells, color-coded for clarity. The visualization is created by an open-source code sample [145]. Brightness is modulated at every point to reflect the distance

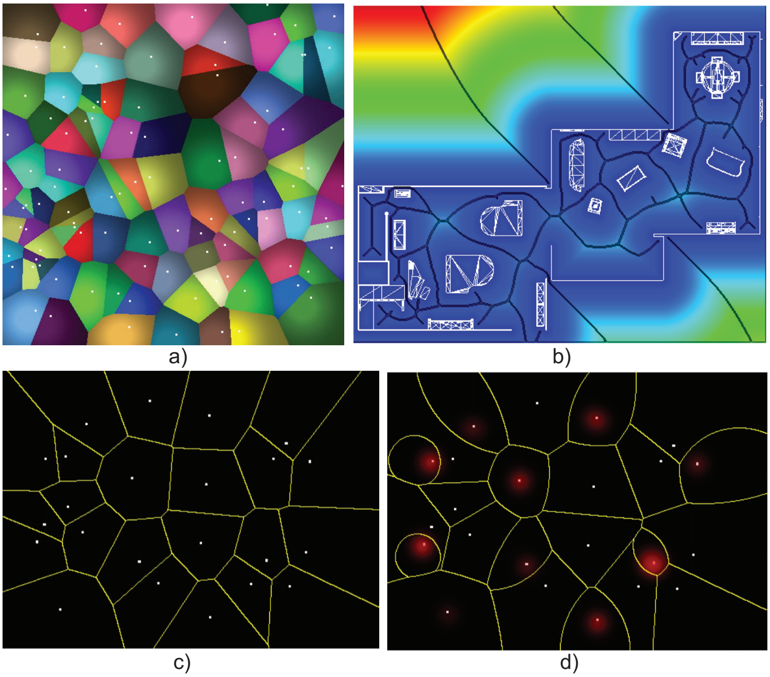


Figure 2.1: Examples of 2D Voronoi diagrams. a) Classical diagram of a point set. b) Generalized diagram of a set of complex sites (in white), with diagram in black and distance field color mapped. c) Classical diagram with its multiplicat-weighted distance counterpart (d).

to the closest site. In detail, let S be the site-set we consider. The function

$$DT_S(\mathbf{x}) = \min_{\mathbf{y}\in S} \|\mathbf{x}-\mathbf{y}\|, \tag{2.12}$$

also called the *distance transform* (DT) of point $\mathbf{y}$, gives the 2D Euclidean distance between from $\mathbf{x}$ to the closest site in S [25]. Note that here $\mathbf{y} \in \mathbb{R}^2$, which is different from y a discrete class label used before. This site is given by the so-called *feature transform* of the set S

$$FT_S(\mathbf{x}) = \arg\min_{\mathbf{y}\in S} \|\mathbf{x}-\mathbf{y}\|. \tag{2.13}$$

The image (a) shows, at each pixel, the values of FT_S color-coded categorically, and the values of DT_S encoded by pseudo-shading. In detail, DT_S is passed through a transfer function to construct an effect similar to diffusely-illuminated equal-radii spheres which are centered at the sites as viewed from above. While not exactly encoding DT_S, every Voronoi cell is thereby rendered as a convex *shaded cushion*, which allows one to easily visually separate adjacent cells. Shaded cushions know a long history in information visualization [155] for the visualization of various types of partitions of, and structures embedded in, 2D space [146, 148, 159]. We shall use variations of shaded cushions in our visualization proposals in Chapter 7. Classical Voronoi diagrams, that is, computed by using standard Euclidean distance to the site-set, can be created by computational geometry methods [15], but also by image-based techniques [60, 142].

However, such diagrams are not directly reflecting the context of kNN classifiers. Indeed, for these classifiers, the actual decision zones would not be the same as the cells of a classical Voronoi diagram, since each such cell corresponds to the influence area of a *single* sample. Rather, we need to consider Voronoi cells that are created by a *collection* of all sites having the same labels. This corresponds to so-called generalized Voronoi diagrams, where sites can be arbitrary collections of points or even higher-order primitives (curves, surfaces) embedded in some space. Figure 2.1(b) shows such a generalized Voronoi diagram. Here, the sites are the structures marked in white, which correspond to curves describing the furniture placed in a building, including the building's walls. The corresponding Voronoi cells, computed by the image-based method in [142], are the curves drawn in black. As visible, these are (far) more complex than the straight lines that delimit Voronoi cells in classical diagrams. A second complication implied by the decision boundaries of kNNs is that k is typically set to values higher than 1. The corresponding Voronoi diagrams created by this extension have, thus, more complex shapes, and a more complex interpretation. Attempts to visualize such diagrams have been made [148]; however, their visual complexity still remains very high.

Voronoi diagrams can be generalized also in other respects besides the definition of their sites. Two well-known generalizations replace the Euclidean distance transform (Eqn. 2.12) by additively-weighted, respectively multiplicatively-weighted versions, where each site has a corresponding weight. These generalizations, known under the names of Johnson-Mehl diagrams and Apollonius diagrams respectively [9], are very relevant in the context of decision boundaries of kNNs, since such classifiers also typically use similar weighting in their construction. Figure 2.1 (d) shows the Apollonius (multiplicatively-weighted) diagram corresponding to the classical Voronoi diagram in Fig. 2.1(c). The green highlights in image (d) give the relative weights of the sites. As visible, this weighting causes cell boundaries to curve, yielding thus more complex shapes.

All above show that, albeit having a simple definition, kNN classifiers can create quite complex decision boundaries. Indeed, to the complexities mentioned above, we should add the fact that kNNs work in *high-dimensional* spaces, whereas all above examples discuss Voronoi diagrams in 2D only. To our knowledge, there is no generic way to compute such generalized Voronoi diagrams in any dimension – let alone to visually explore them. Hence, understanding the boundaries of such classifiers is clearly challenging.

For kNN classifiers, k is the only user defined hyperparameter. The decision boundaries induced by the method will be highly dependent the chosen k. In one extreme setting, for $k = 1$, a model very susceptible to errors due to noise or outliers will be inferred, while for larger values of k smoother decision boundaries will be formed. For too large k values, however, important information in the data might be ignored as small data clusters might be ignored. Cross validation, as mentioned earlier in this chapter, can be employed to assist the selection of this hyperparameter.

The computational costs of kNN can be relatively high as distance computations for large datasets of high dimensional data is expensive. However, acceleration techniques exist here, which compute the *approximate* nearest neighbors (ANNs) [7]. Conceptually speaking, such techniques trade off a small user controlled tolerance ϵ when evaluating Eqns. 2.12 and 2.13. This allows them to effectively partition the high-dimensional space into hierarchical structures such as BSP trees or kd-trees. These can next be searched top-down to yield the k nearest neighbors very efficiently. This way, while formally speaking, the kNN learning method has no “training” phase, the training costs can be thought of being the costs required to build the search structures for ANN. For more information of efficient k nearest neighbors search structures, we refer to recent surveys [156].

kNN is a simple yet very powerful classification method for smooth target functions, which same class points are clustered together. Arguably the most common distance metric used with this method is the Euclidean distance. Depending on the data, other similarity measures might perform better, as they better reflect data points relations in original data space. As discussed in this section, kNN decision boundaries have intimate connections to Voronoi diagrams and, for simple cases, can also be visualized this way. Given these aspects, we included kNN in the set of classifier techniques we study next. In Chapter 7, we will explore further the relation between distance transforms, feature transforms, and the decision boundaries of such classifiers.

2.1.4 *Random Forests*

Decision Trees (DT) are another type of nonparametric classifiers. The classification rule employed by this model consists in traversing a tree from the root to a leaf, where the class labels are stored. During the traversal, each intermediate node, *i.e.* non-leaf nodes, in the path splits input space at a certain feature according to a *rule* the controls what is the next node to follow on the path. Arguably, the most common rule for real valued features consists in *thresholding*, such as $[\![x_i^j < t]\!]$, where t is the threshold, i is the index of the data point, and j is the index of a feature in a data point vector $\mathbf{x}$. Another common rule, but for binary (or categorical) features would be $[\![x_i^j = 1]\!]$. As $\mathbf{x} \in \mathbb{R}^d$, this kind of rule is in fact partitioning input space along dimension i. Each path from the root to a leaf node defines a unique region of data space, thus the number of regions created by the DT method are equal to the number of leaves in the tree. Trees with many leaves may overfit to the data as the thresholding rules will be in fact memorizing the dataset and, in order to avoid that, it is important to penalize large trees.

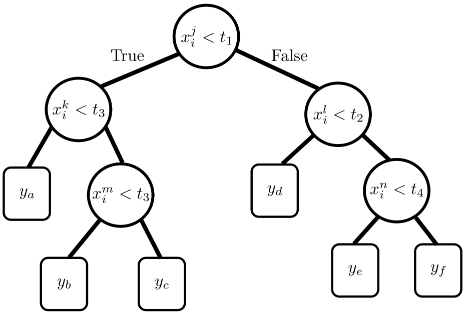


Figure 2.2: Simple representation of a Decision Tree. Internal nodes, which perform a decision, are represented by circles and leaf nodes, which hold labels, are drawn as squares. A traversal in this tree moves to the left if the condition is satisfied, or to the right child if it is not.

As an optimal tree construction algorithm is unfeasible, several heuristics, *e.g.* greedy search, are used that lead to acceptable performances [129]. One example of such algorithms is Iterative Dicotomizer 3 (ID3), described in Algorithm 1, which seeks to repeatedly split the dataset along the feature that has the most *information gain* (or *entropy*), that will be reviewed after the algorithm presentation. ID3 is a recursive algorithm that stops when either all samples in the dataset have the same label, every feature has been split or there are no more samples with respect to which split the dataset.

Information gain function can be computed using *Shannon entropy*: $E(\mathcal{Y}) = \sum_{y \in \mathcal{Y}} -p(y) \log p(y)$, where $p(y)$ is the probability of label y. The idea is to measure the difference in entropy of the label distribution after a certain feature is removed:

$$G(\mathcal{X}, \mathcal{Y}, i) = E(\mathcal{Y}) - \sum_{v} P(x_i^j = v) E(\mathcal{Y} | f_i = v).$$

$P(x_i^j = v)$ is computed by dividing the number of samples for which the jth feature is equal to v by the total number of samples.

Due to the stop conditions defined, the algorithm above usually build large trees, which leads to overfitting. To avoid overfitting a common strategy is to *prune* the induced trees, reducing its size, traversing the

Algorithm 1 ID3($\mathcal{D}$, A)

```
Input: Dataset $\mathcal{D}$, index set of features $A \subseteq [n]$
Output: Node $n$ of the tree
  if $y_i = y_j \forall y_i, y_j \in \mathcal{Y}$ then
    return leaf node $n$ with label $y_i$
  end if
  if $A = \varnothing$ then
    return leaf node $n$ with label $\text{argmax}_{k \in C} \sum_{\mathrm{y}_i \in \mathcal{Y}} [\![\mathrm{y}_i = k]\!]$
  end if
  if $\mathcal{D} = \varnothing$ then
    return leaf node $n$ with same label as most common class in parent node.
  end if
  $G = \text{InformationGain}(\mathcal{X}_A)$ {//Find feature that maximizes information gain}
  $i = \underset{i}{\arg\max} G$ {//Split dataset at that feature}
  $\mathcal{D}_l = \{(\mathbf{x}_j^i, y_j) \in \mathcal{D}, \mathbf{x}_j^i < t\}$
  $\mathcal{D}_r = \{(\mathbf{x}_j^i, y_j) \in \mathcal{D}, \mathbf{x}_j^i \geq t\}$
  $A' = \{A \setminus \{i\}\}$
  New node $n$
  Left child of $n$ is $n_l = \text{ID3}(\mathcal{D}_l, A')$
  Right child of $n$ is $n_r = \text{ID3}(\mathcal{D}_r, A')$
  return n
```

tree multiple times starting from the leaves to the root, merging nodes that would not affect the generalization error.

Random Forests is a classifier that consists in combining a set of Decision Trees. The classification rule outputs the label returned by the majority of the individual trees when queried to label a data point, as follows

$$h(\mathbf{x}) = \text{argmax}_{k \in C} \sum_{i}^{M} [\![\text{DT}_i(\mathbf{x}) = k]\!],$$

where M is the number of Decision Trees employed and C is the list of all possible labels.

In order for the trees to induce different models, a common strategy, known as *bagging* in the literature, is to build them using different training sets, randomly sampled from the original input dataset, and also different possible lists of available features for each of those sets. In this strategy, different trees will be constructed using different data points while also taking into consideration different features for inference.

While it is easy to visualize and explain the decision process of a single Decision Tree, it might be challenging to understand the final *ensemble* of classifiers, which is the case for Random Forests. Hence this model belongs to a potentially interesting area of classifiers to be inquired by the visualization methods.

2.1.5 *Neural Networks*

Neural Network models are conceptually inspired from the functioning of a biological human brain and are based on the combination of *neurons* as basic computing units. Analogous to their biological counterparts, the artificial neuron (the so-called *perceptron*) receives *stimulus* (inputs) and might fire a reponse (output) according to its inner workings (activation function).

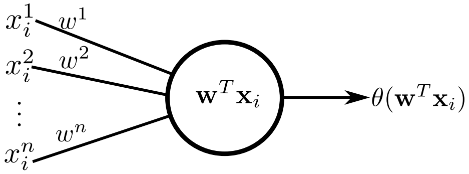

x_i^1
w^1
x_i^2
w^2
$\vdots$
w^n
x_i^n
$\mathbf{w}^T\mathbf{x}_i$
$\theta(\mathbf{w}^T\mathbf{x}_i)$

Figure 2.3: Representation of a perceptron. An input vector $\mathbf{x}_i$ is linearly combined with a weight vector $\mathbf{w}$ and the output is return by a funtion θ.

Figure 2.3 shows a basic neuron that outputs a signal $y = \theta(\sum w^j * x_i^j) = \theta(\mathbf{w}^T\mathbf{x}_i)$, where θ is an activation function[1]. Here, $\mathbf{x} = \{x^1, \ldots, x^n\}$ is the n-dimensional vector containing the inputs x^j of the neuron, and $\mathbf{w} = \{w^1, \ldots, w^n\}$ is a corresponding weight vector. In its basic form, the perceptron computes a linear model, as the Logistic Regression discussed in Sec. 2.1.1 selecting θ as the logistic function, or even the basic linear model presented in Eqn. 2.1. The classification power of a single perceptron is small, as it is a simple linear combination of parameters and inputs. However, perceptrons are capable of inducing complicated and flexible decision boundaries when combined into a computational graph (or network) in which the output of a given neuron becomes the input of one or several other neurons.

At a basic level, perceptrons can be combined into a so-called *Multi Layer Perceptron* (MLP), a model in which neurons are stacked into layers as in Fig. 2.4. An input data point $\mathbf{x}$ is said to be the *input layer*. The last layer is called the *output layer*. All layers between input and output are known as *hidden layers*. The decision rule is computed by feeding the output of a given layer as input to the next, in a procedure known as *forward pass*. As the number of neurons in each layer vary and based on the choice of the activation function, each layer effectively performs subsequent data transformations, even modifying the feature vector dimension in the path up to output layer.

---

[1] As with Logistic Regression, we assume that $\mathbf{x}$ is a bias augmented vector.

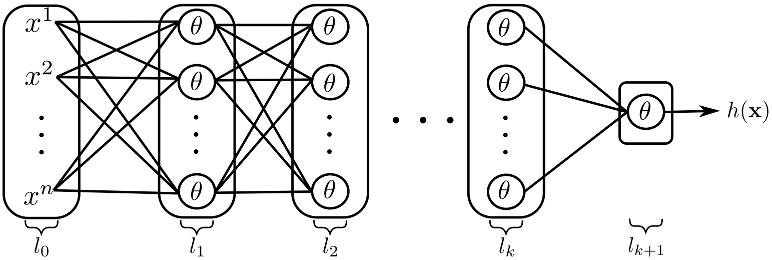


Figure 2.4: Graph representation of the computation carried out by a Multi Layer Perceptron. Input layer l_0 is a data point $\mathbf{x}$, followed by k hidden layers $l_1, \ldots, l_k$, and an output layer is l_{k+1}, which returns the classification decision.

The forward pass, *i.e.*, the decision rule, of a MLP model can be efficiently computed by consecutive matrix-vector multiplications. At a certain layer l, let $\mathbf{x}^{l-1}$ be the input vector to layer l of dimension $d^{l-1}+1$, that is, the bias-augmented output of the previous layer $(l-1)$. Let W^l be a matrix of dimensions $d^{l-1}+1 \times d^l$, where d^l is the number of neurons in l. W^l represents the weights that connect the outputs of layer $l-1$ to every unit in layer l, as the rows of W represent a single output unit from $l-1$ (plus another row for the bias term) and the columns of W are the neurons in l. With these notations, the output $\mathbf{o}^l$ of layer l is given by

$$\mathbf{o}^l = \theta\left(\left(W^l\right)^T \mathbf{x}^{l-1}\right).$$

The bias-augmented input to the next layer $l+1$ is then

$$\mathbf{x}^{l+1} = \begin{bmatrix} 1 \\ \mathbf{o}^l \end{bmatrix},$$

hence, a MLP that is formed by k hidden layers, as in Fig.2.4, computes a decision rule $h(\mathbf{x}) = L_{k+1}(\ldots(L_2(L_1(x)))) = y$. We can see the set of all parameters of this model in a vectorized fashion as $\mathbf{w} = W^1, W^2, \ldots, W^{k+1}$. This notation allows us to compactly, and in accord to previous sections, refer to an error function for this model as E.

A typical error function used to train different Machine Learning models, which is well suited for MLP, is the *mean squared error* (MSE), defined as

$$E(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(h(\mathbf{x}) - y\right)^2. \tag{2.14}$$

Similarly to the other methods discussed previously, the training algorithm searches for the parameters $\mathbf{w}$ that minimize Eqn. 2.14 by gradient based optimization. However, different from Logistic Regression

(LR) or SVM, the minimum possible error, *i.e.* global minimum, for a given dataset is not guaranteed to be found as $E(\mathbf{w})$ in Eqn. 2.14 is not a *convex* function [24]. Hence, gradient based optmization methods will likely find a set of parameters that lead to a local minimum for Neural Networks, while the same methods can find the global minimum error for LR and SVM due to the convex nature of the error functions employed. However, in practice, this is not usually a problem, as neural network training algorithms can find weights that achieve reasonable performance.

Backpropagation is the algorithm used to update the weights of a neural network during the training phase. As the output of a single perceptron depends on the output of the layer behind it, the backpropagation algorithm consists in computing the partial derivatives of the errors with regard to each weight by successively applying the chain rule. The idea behind this algorithm is to compute the partial derivatives starting from the output layer back to the input, performing a backward pass that computes the partial derivatives of a layer l using those of layer $l + 1$.

Besides MLP, other layouts (also called *architectures*) are possible with Neural Networks. In particular, the so-called *Convolutional Neural Networks* (CNNs) are composed of convolutional layers, max pooling layers, and regular fully connected layers. This type of architecture achieved breakthrough performance on diverse computer vision tasks and is now the standard tool for image classification problems [73, 76].

A convolutional layer is composed of stacked *masks* that apply convolutions on the layer's input to output a “filtered signal” to the next layer. In a simplified setting, the parameters of a convolutional layer are the number of filters K, their size F, stride S and the amount of zero-padding P. Stride controls how many units the mask will move when sliding on the input volume, thus a stride of 1 will visit every input (*e.g.*, pixel of an image), while a value of 2 of this parameter will skip one pixel as the mask moves. The zero-padding parameter sets how many pixels are added to the borders of the input dataset (with a value of 0). Without zero-padding, the output is necessarily smaller than the input since it is not possible to center a mask over every pixel. F determines the width and height of the filters, while their depth must necessarily match the depth of the input to the layer.

The input to a convolutional layer is a volume, such as an image of dimensions $W \times H \times C$, where W is the width, H is the height, and C is the number of channels, *e.g.*, 1 for grayscale images or 3 for color (*e.g.*, RGB) images, respectively. The output of a convolutional layer is a volume of dimensions $W' \times H' \times C'$, which is the transformed input layer by each of the K filters stacked together. The dimensions of the output volume can be computed as

$$\begin{aligned} W' &= (W - F + 2P)/S + 1; \\ H' &= (H - F + 2P)/S + 1; \\ C' &= K. \end{aligned}$$

A convolution is computed between the input and the weights of a filter. For a single input coordinate (i, j) of the image (or volume) space, the output of a convolution is

$$g(w, h) = \sum_{l=1}^{K} \sum_{di=-F}^{F} \sum_{dj=-F}^{F} I_{i+di, j+dj, l} M_{di, dj, l}, \tag{2.15}$$

where M is a certain mask and I is the input volume. Note that as the input and the filter have the same depth, we do not consider variations across that dimension. A full convolution is the result of the application of Eqn. 2.15 to every coordinate (i, j) of the input volume.

A max pooling layer is usually placed after a convolutional layer, or a sequence of convolutional layers, and its objective is to reduce the size of the feature maps. The parameters of a max pooling layer are its size F and the stride S, similar to the parameters of a convolutional layer. In this type of layer, a mask of size $F \times F$ is applied to each slice of the volume along its depth, *i.e.* number of filters K in the previous convolutional layer. As the mask defined by the max pooling layer slides over a slice of size $W \times H$, only the maximum value among the $F \times F$ features under the mask is kept to the next layer, discarding the remaining. Notice that a max pooling layer does not contain any weight that will be learned during the training phase of the network.

A depiction of the data transformation performed by the layers of CNN is shown in Fig. 2.5. In this example, the input is a (grayscale) image of a handwritten digit from the MNIST dataset [75]. Data is transformed by consecutive layers until it is linearized into a feature vector. From that point onward, regular fully connected layers can be applied to compute a decision rule and output a data label in a classification scenario.

In the context of our work, neural networks are arguably the most interesting ML model to study. The reason is two-fold: (a) they are state of the art methods that deliver high-quality results for many challenging classification and regression problems; and (b) these models are, in general, typically seen as “black boxes”, due to the lack of interpretability of their decision function. Opening this black box is seen as highly valuable in a wide range of fields [10, 35, 130, 170].

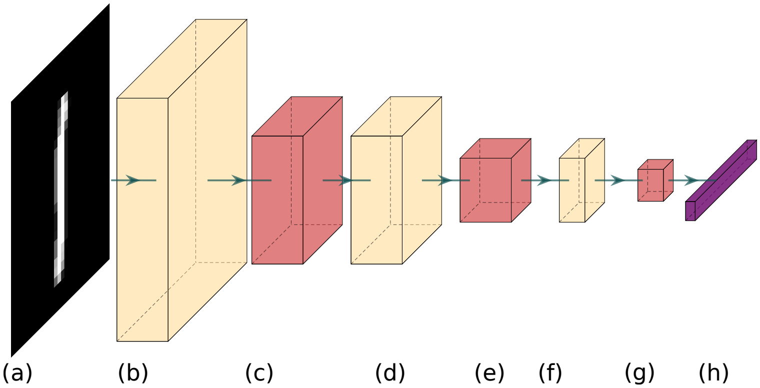


Figure 2.5: Data transformation performed by a CNN. Yellow volumes represent the output of a convolutional layer. Red volumes are the output of max-pooling layers. In this example, the network takes a grayscale image as input (a) and the first convolutional layer transforms it into a volume of stacked filtered signals (b). After that, a max-pooling layer reduces the size of the volume along two dimensions and outputs the new volume in (c). The sequence of convolutional and max-pooling layers continue to perform data transformations (d) - (g) – until the volume is "linearized" in (h).

2.2 VISUAL ANALYTICS FOR MACHINE LEARNING

Information visualization has been largely applied to the visualization and understanding of high-dimensional data. As such, it relates to our context in two different ways. First, since machine learning inherently deals with *high-dimensional data*, visualization methods that generically enable users to see the structure of such data and reason about it, are of evident interest. Secondly, more specific techniques have been proposed in the visual analytics context for explaining and improving *classifiers*. At a high level, the second class of techniques can be seen as relying upon, but also specializing, the first class.

We structure this discussion as follows. First, we briefly overview in Sec. 2.2.1 a number of general-purpose information visualization techniques for high-dimensional data. While these techniques are used in some applications related to machine learning, they are not the first or most-encountered candidates for such tasks in practice. We explain their limitations, and based on these, our reasons for not subsequently considering them in our work. Section 2.2.2 introduces dimensionality reduction techniques. These have several key advantages when dealing with high-dimensional data, and thus form the mainstream of visualization approaches used in machine learning. As such, we also chose to base our visualization work next on techniques in this class. We explain further in this section a few of the techniques in this class that we further consider in our work. Section 2.2.3 presents a separate class of techniques – inverse projections – which perform the inverse operation to dimensionality reduction. We outline several salient examples of these techniques and explain how they can contribute to our visualization goals. Finally, Section 2.2.4 presents specialized visual analytics techniques and tools designed to support tasks in machine learning such as explaining and improving classifiers.

2.2.1 *High-Dimensional Data Visualization*

In the preliminaries definitions for Machine Learning in Sec. 2.1, we defined a high dimensional dataset $\mathcal{D}$ to be composed of pairs of observations ($\mathbf{x}_i$) and labels (y_i). For visualization purposes, we refer to $\mathcal{X}$ as our dataset of interest, and the definitions described earlier still apply.

High-dimensional data visualization is a subdomain of information visualization (Infovis) which aims at creating visual depictions of such high-dimensional datasets $\mathcal{X}$ [68, 80, 90, 150]. In general, and within suitable simplifications required by a brief presentation, such techniques can be classified between two extremes, as follows. At one extreme, *dimension-centric* techniques aim to explicitly encode the dimensions $\mathbf{X}^j$ of $\mathcal{X}$ in the available visual space. Thereby, such techniques support well tasks and questions which are intrinsically related to dimensions, such as finding (groups of) strongly correlated dimensions, seeing how the sample values change along a given dimension, and finding extremal or outlier values of these values along one or more dimensions. However, reasoning about specific observations is not (always) easy with such techniques, since the observations are not (saliently) displayed by the visual representation. A very simple example of such a technique would be a correlation matrix showing, for each dimension-pair $(i, j) \in [1, n] \times [1, n]$ the Pearson correlation of $\mathbf{X}^i$ and $\mathbf{X}^j$. We can see these correlations explicitly, but we do not know how the observations contribute to them. At the other extreme, *observation-centric* techniques do the opposite: They explicitly encode the observations $\mathbf{x}_i$ of $\mathcal{X}$ in the visual representation, but leave little space for explaining their dimensions. These techniques, thus, support well tasks where reasoning about a specific observation is important. One simple example hereof is a classical Excel table view: In this table, we can search for a given observation (row) and fully see all its details. However, we cannot (easily) reason about entire columns (dimensions).

In practice, high-dimensional visualization techniques are always in between the above-mentioned two extremes. We next discuss three of the most used, and best known, such techniques.

Table lenses [113] generalize the simple idea of Excel (or similar) table views. Figure 2.6(a, left) shows such a table view in which several tens of rows (observations) are displayed, each having seven attributes

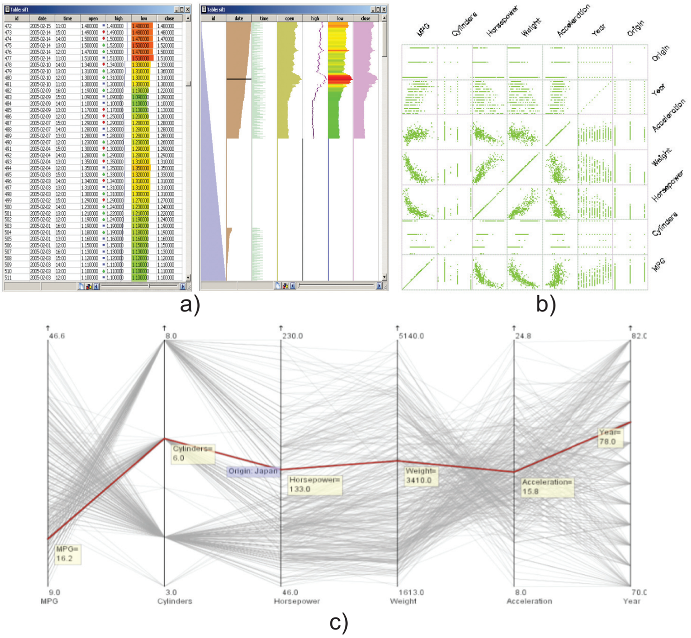


Figure 2.6: Examples of high-dimensional visualizations. a) Table view and its corresponding zoomed-out table lens. b) Scatterplot matrix. c) Parallel coordinates plot.

(columns). As explained above, such a detailed table view allows one to precisely reason about the observations, but not much more. Figure 2.6(a, right) shows the underlying idea of a table lens. Simply put, this is a zoomed-out view of the detailed view, where each row is shrunk to become a single horizontal pixel line. In this design, cell text is no longer visible. However, the actual cell values can be mapped to colors and/or bar lengths. This way, essentially each column is reduced to a line or bar chart. Hence, users can reason about entire dimensions at a time, by seeing trends in the respective charts. For example, in Figure 2.6(a, right), we can easily see that several columns show very similar charts, thus have strongly correlated attributes. Conversely, the leftmost two columns show opposite-pattern charts, thus, indicate inversely correlated attributes. The "lens" idea comes in next, by allowing users to select a specific point (row) in the charts, upon which rows close to that point are rendered atop the zoomed-out view to show details on demand. Table lenses scale very well to datasets of hundreds of thousands of observations by using suitable aggregations of contiguous table rows that would have otherwise subpixel size [149]. However, the number of dimensions (columns) this technique can handle is limited to roughly 10..20. Moreover, table lenses cannot easily show more complex data patterns such as groups of samples (rows) that are similar.

Scatterplot matrices [21], or SPLOMs, are closer to a dimension-centric technique than table lenses. Simply put these use a small-multiple design that creates a table of scatterplots, one for each pair of dimensions $(\mathbf{X}^i, \mathbf{X}^j$, with $1 \leq i \leq n, 1 \leq j \leq n$. Figure 2.6b shows such a scatterplot matrix for a $n = 7$-dimensional dataset. The table allows one to see how each such dimension-pair is correlated (or not), but also whether different dimension-pairs exhibit similar correlation patterns. Note that the displayed matrix is symmetric by definition so, strictly speaking, displaying only its upper-triangular half, or the other half, would be enough. However, SPLOMs are not very effective when supporting observation-centric tasks. An observation, actually, does not even have a clear visual identity in this metaphor, as it is represented by a set of n^2 points, one located within each SPLOM cell. While SPLOMs have been further enhanced by various interaction and visualization mechanisms [117, 163], their usage can become cumbersome when the number of dimensions exceeds roughly $d = 10$. This poses a clear limitation in the context of our work.

Scagnostics [154, 160] recognize the aforementioned problem of SPLOMs and aim to address it, for large n, by explicitly searching, among the set of all possible n^2 scatterplots, for a small subset of "interesting" ones. Upon finding such plots, they are presented to the user for further inspection. Interest can be defined, and computed, by using many types of metrics, which essentially search for specific patterns in the scatterplot, related to specific tasks that the user is interested in [29, 77, 99]. Simple patterns include distributions close to lines (showing thus strong direct or inverse correlations), clumpiness (showing the presence of well-separated data clusters), or the presence of outliers. However, such techniques have several limitations. One must, in general, compute all n^2 possible scatterplots to search for interesting patterns, which is expensive. The detection of patterns is also far from being a trivial process. More importantly in our context, finding patterns which are spanned by more than a few dimensions, but do not appear in any of the corresponding two-dimensional scatterplots, is problematic [137]: If we are to search for such patterns, the search complexity explodes; and even if we can find them, how to show them using scatterplots only? Several additional techniques related to scatterplots, and thus scagnostics exist. Principal curves and their variations allow simplifying a scatterplot distribution to a set of curves, thereby making depiction more compact and also allowing to search for structural patterns in the data [47, 56, 97]. Summarizing the above, and especially given the fact that we do not know which exact patterns we are looking for in the high-dimensional data of our machine learning applications (to display decision boundaries), we do not consider scagnostic techniques further.

Parallel coordinate plots [64], or PCPs, are the fourth and last type of technique we discuss here. PCPs use a similar layout to table lenses (see Fig. 2.6d): They create n vertical axes, one per dimension, and use the values of x_i^j to place points along these axes for all samples. Next, they connect all values x_i^j for a given sample $\mathbf{x}_i$ to depict this sample as a polyline. The overall design allows several analyses. First, one can look at each axis j to see the distribution of values along $\mathbf{X}^j$, or compare several such axes to reason about their distribution differences. More interestingly, the rendered polylines show patterns describing relations between dimensions. For instance, an X-like pattern, as visible in Fig. 2.6d between the leftmost two axes, indicates a strong inverse correlation. Parallel-line patterns, such as between the two axes in the middle of the image, indicate a strong direct correlation. In contrast to SPLOMs, reasoning about specific observations is easier in PCPs, as these can be directly selected or brushed over to examine them – such as the observation marked in red in the figure. Despite these features, PCPs also have limitations. Just as table lenses, they cannot show more than roughly 10..20 dimensions. Moreover, ordering dimensions is necessary to be able to examine correlation patterns between adjoint dimensions in the visualization. Finally, PCPs can easily create clutter since every observation takes significant screen space. Given our work context, we also do not see how PCPs could be leveraged to reason about patterns formed by multiple observations together, beyond those such a simple correlations and similar, and thus we do not consider PCPs further.

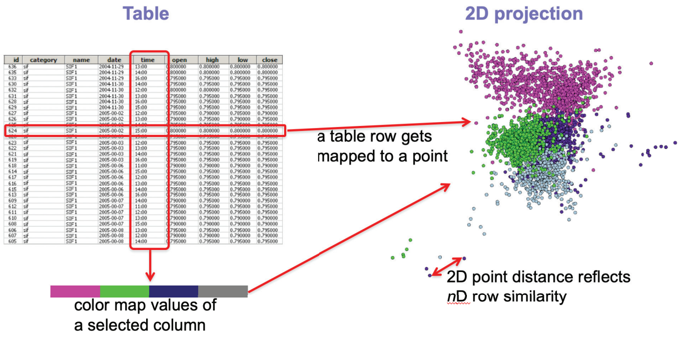


Table
2D projection
id
category
name
date
time
open
high
low
close
a table row gets mapped to a point
color map values of a selected column
2D point distance reflects nD row similarity

Figure 2.7: Using projections in machine learning. From a high-dimensional dataset (left), a projection (right) is created using the independent data attributes (features). Next, one can color map the dependent data attribute (class label) to see how the features succeed (or not) to predict it.

2.2.2 *Dimensionality Reduction*

Interesting problems often involve high-dimensional data, such as image recognition (where every pixel represents a value along one dimension) or medical task related to genomic data, where millions of combinations are possible. Even trivial tasks, such as hand written digit recognition (MNIST [75]) from 28 × 28 pixels grayscale images already requires the handling of 784-dimensional data.

The *curse of dimensionality* is an important concept for every task that handles data in high-dimensions. As the number of dimensions grow, our human intuitions are no longer (fully) valid, since a finite set of samples can only sparsely cover a high-dimensional data space, as its volume grows exponentially with the number of dimensions n. Moreover, the computational power needed to train ML models also increases with the number of dimensions. As the complexity of many algorithms depends on the number of dimensions, high dimensional data also poses a problem due to scalability.

Although most real problems are high-dimensional, the samples are often restricted to a small region of data space. Consider the example of MNIST dataset cited above, where the set of samples are contained in a 784-dimensional space, *i.e.* $x_i \in 255^{784}$, only a small fraction of all the possible points in this space are valid digits.

One approach to handle such a scenario is through the use of *dimensionality reduction* techniques. The basic idea is to transform data from a high dimensional space into a lower-dimensional one, while keeping important properties from the original space in the new space. Formally put, given a set of N n-dimensional samples $\mathcal{X}$, we want to create an identical-size set $P(\mathcal{X}) = \{\mathbf{y}_i\}$, $1 \leq i \leq N$, where each sample $\mathbf{y}_i \in \mathbb{R}^m$ is embedded within a (far) lower dimensional space, that is, $m \ll n$. In the above process, it is very important to mention that there is a one-to-one correspondence of original and low-dimensionality samples, *i.e.*, $\mathbf{y}_i$ corresponds to $\mathbf{x}_i$. This can be denoted by writing $\mathbf{y}_i = P(\mathbf{x}_i)$. When referring to a dimensionality reduction context, we denote the set $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\}$ as the set of samples in a lower dimensional space, different from Sec. 2.1 where $\mathcal{Y}$ was a set of labels. With that we aim at preserving the notation used most commonly by dimensionality reduction community.

Here, P can be seen as the dimensionality-reduction (DR) operation, also called sometimes *multidimensional projection*. Note, however, that this notation does not imply that the projection of $\mathbf{y}_i$ can be solely computed by knowing $\mathbf{x}_i$. Rather, we can speak, in functional terms of the entire dataset $P(\mathcal{X})$ being computed by projecting the entire dataset $\mathcal{X}$.

Making inferences in this new, low-dimensional, space should be easier as the dimensionality is lower, since computing various characteristics of the data is easier. A particular instance of this statement refers to *visualization*. Indeed, when $m \in \{2, 3\}$, we can directly draw $P(\mathcal{X})$, *e.g.*, as a scatterplot. Then, by visualizing this scatterplot, we can find (hopefully) patterns which exist in the high-dimensional space and the projection operator P managed to preserve.

In our context, projections are particularly interesting instruments. Figure 2.7 illustrates this schematically. Consider a high-dimensional dataset, represented by the table left in the image. Assume this table has

$n+1$ columns, where n columns represent the data features, and the final column represents the class labels, such as present in a training or test set. The n feature vectors can be used to generate a projection, such as shown on the right. Assuming that the projection technique P used for this will preserve data structure – which typically means that samples which have similar feature vectors are projected close to each other – the projection can help us with two main tasks. First, if we see *groups*, or clusters, of points forming in the projection, it means that the data is not uniformly distributed in the high dimensions, but rather consists of sets of clusters too. This is already an important unsupervised machine learning insight: Indeed, if the data (as represented by its feature vectors) were uniformly distributed in this space, then it would not be likely that we can *use* the respective features to "split" the data into different classes. Secondly, we can color the projection by the values of the class attribute (column). If we, next, see that clusters of points (in the projection) have the same color, it means that the respective samples, which are similar given their feature vectors, are of the same class, and hence the feature vectors are likely suitable to *predict* the class attribute. Conversely, if we see that such clusters consist of a mix of colors, the respective features may have trouble in building a good classification model. By extension, seeing a few "outlier" points – having one color but surrounded by many points of a different color – indicates us potential classification problems. Finally, plotting the actual misclassified points atop of such a projection allows us to reason about which data attributes these have and how these may have caused problems to the classifier. These, and other, scenarios have been recently examined in recent literature [114–116]. In particular, Rauber *et al.* [115] show that projections can be used as good *predictors* for the ease of constructing a good classifier from a given training set. The idea is further developed in [13], who show how projections can be used to *improve* an existing classifier by semi-supervised training.

Hence, given all above observations, we deem projections to be the most suitable method for further exploring for our research goals. Also, at this point, we can further connect our specific visualization aims – depicting decision zones – to the projection metaphor. Consider again Fig. 2.7. If, as in that figure, we observe that our machine learning dataset projects into a set of well-separated clusters, and coloring these by the class labels *inferred by a trained classifier* shows compact same-color point groups, it is clear that the decision boundaries fall somewhere *in between* these colored groups. However, projections – depicted as class-label-colored scatterplots – do not *explicitly* show where such boundaries occur, leaving this task to the intuition, and largely imagination, of the user to place them in the blank space between points. As we shall see starting from Chapter 4, our goal will be to construct and explicitly visualize such decision boundaries atop of, and using, projection scatterplots.

In the past decades, tens of projection techniques have been proposed. These are discussed in detail in several surveys [27, 36, 48, 61, 83, 91, 110, 140, 164]. These surveys have emerged from various fields, such as data science, statistics, machine learning, and information visualization. As such, they cover different aspects, such as proposing taxonomies to classify projection techniques according to their underlying algorithms and models; ways to define and measure the errors created by projections; types of patterns that specific projection techniques are good at capturing; and assumptions about the input high-dimensional data these techniques expect. It is impossible, and arguably of little use to summarize these findings and relate them to our concrete research context.

Rather, for choosing which dimensionality reduction we will next use in our work, we will consider a separate recent survey [37]. In this survey, the authors analyze 44 actual implementations of projection techniques against 20 datasets, using six different quality metrics from the literature. The presented experiments, done by performing extensive grid searches over the projections' hyperparameters, provide several important insights in the quality, and computational complexity, of the respective algorithms. Following their analysis, we selected next three projection techniques to consider in our research work. These are described next.

2.2.2.1 *LAMP: Local Affine Multidimensional Projection*

Local Affine Multidimensional Projection (LAMP) [66] is a parametric dimensionality reduction method that aims at preserving in the lower dimensional embedding the Euclidean distance observed between the points in the original, high dimensional, data space.

To accomplish this, LAMP algorithm constructs an affine mapping for each of the high-dimensional data points $\mathbf{x}_i$ by using a set of so-called *control points*. Control points are a small subset of the entire dataset that is already be projected. More formally, we denote the set of control points as $\mathcal{X}_S = \{\mathbf{x}_1^S, \ldots, \mathbf{x}_k^S\}$, $\mathcal{X}_S \subset \mathcal{X}$. Let $\mathcal{Y}_S = \{\mathbf{y}_1^S, \ldots, \mathbf{y}_k^S\}$, $\mathcal{Y}_S \subset \mathbb{R}^2$, be the projections of $\mathcal{X}_S$. These projections can be constructed by any suitable method, be it one of the multidimensional projection techniques known in the literature, or even by having the user manually place the points in $\mathcal{Y}_S$ in the 2D space to reflect perceived similarities. The key idea of LAMP is that constructing a projection for the control point set $\mathcal{X}_S$ is much easier (and/or faster) than constructing a projection for the typically far larger dataset $\mathcal{X}$. Hence, LAMP "extrapolates" the control-point projection $\mathcal{Y}_S$ to project the entire dataset $\mathcal{X}$, by an affine mapping.

For every point $\mathbf{x}$, LAMP defines an affine mapping of the form $f_{\mathbf{x}}(\mathbf{p}) = \mathbf{p}M + \mathbf{t}$. The matrix M and the vector $\mathbf{t}$ are found by solving the following optimization problem

$$\begin{aligned} \underset{M,\mathbf{t}}{\text{minimize}} \quad & \sum_{i=1}^{k} \alpha_i \left\| f_{\mathbf{x}}(\mathbf{x}_i^S) - \mathbf{y}_i^S) \right\|^2 \\ \text{subject to} \quad & M^T M = I \\ \text{with} \quad & \alpha_i = \frac{1}{\left\| \mathbf{x}_i^S - \mathbf{x} \right\|^2}. \end{aligned} \tag{2.16}$$

Hence, for each point $\mathbf{x}$ to be projected, the objective of LAMP is to find a mapping that best matches the projection done for the set control points $\mathcal{X}_S$, weighted by the distances from $\mathbf{x}$ to each control point.

As shown in the original publication [66], the minimization problem in Eqn. 2.16 is equivalent to the minimization problem below

$$\begin{aligned} \underset{M}{\text{minimize}} \quad & \sum_{i=1}^{k} \alpha_i \left\| \hat{\mathbf{x}}_i M - \hat{\mathbf{y}}_i \right\|^2 \\ \text{subject to} \quad & M^T M = I \\ \text{with} \quad & \hat{\mathbf{x}}_i = \mathbf{x}_i - \frac{\sum_{i=1}^{k} \alpha_i \mathbf{x}_i}{\sum_{i=1}^{k} \alpha_i}, \\ & \hat{\mathbf{y}}_i = y_i - \frac{\sum_{i=1}^{k} \alpha_i \mathbf{y}_i}{\sum_{i=1}^{k} \alpha_i}. \end{aligned} \tag{2.17}$$

After M is found by solving the optimization problem in Eqn. 2.17, the projection of a given point is obtained as

$$P(\mathbf{x}) = \left(\mathbf{x} - \frac{\sum_{i=1}^{k} \alpha_i \mathbf{x}_i}{\sum_{i=1}^{k} \alpha_i} \right) M + \frac{\sum_{i=1}^{k} \alpha_i \mathbf{y}_i}{\sum_{i=0}^{k} \alpha_i}. \tag{2.18}$$

LAMP can be modified to work as a *local* projection method by restricting the number of points from the control set $\mathcal{X}_S$ used when inducing M. For this, a common choice is to consider in Eqn. 2.18 only a (small) number of the nearest control points to the current point $\mathbf{x}$ to project.

LAMP is an easy to implement and relatively fast method, as the minimization problem can be solved by employing Singular Value Decomposition (SVD). However, the fact that LAMP relies on the projection of a control point set can be a problem. If the control point set $\mathcal{X}_S$ does not describe well the total dataset $\mathcal{X}$ to be projected, then LAMP may create a poor quality projection. This can happen, for example, when $\mathcal{X}$ samples a much larger region of the high-dimensional data space than $\mathcal{X}_S$. This issue is not unique to LAMP; all other projection methods based on control points (or landmarks) suffer from the same limitation [22, 91, 132]. Interestingly, recent projection methods based on deep learning also exhibit the same limitation [40]. The analogy here is that a learning algorithm can extrapolate the information learned from a training set only up to a maximal "distance" to this training set.

A second important ingredient of LAMP is the assumption of an inverse projection method, *i.e.*, a mapping from a point $\mathbf{y} \in \mathbb{R}^2$ to a point $\mathbf{x} \in \mathbb{R}^n$ that would project at (or close to) $\mathbf{y}$. We discuss this inverse projection separately in Sec. 2.2.3.1, as inverse projections will be central to our own work in visualizing decision zones.

2.2.2.2 *t-SNE: t-Distributed Stochastic Neighbor Embedding*

t-Distributed Stochastic Neighbor Embedding (t-SNE) [82] is a nonparametric, nonlinear projection technique based on minimizing the difference between the distributions of data points similarity in high and low dimensions. That is, similarities inferred for the 2D space must be as close as possible to that of the original *nD* data space. This method is popular and frequently employed for the visualization of high-dimensional datasets due to its underlying neighborhood preservation property, that is, it is likely that neighbor points in *nD* will be projected closely in 2D. As a consequence, if one has a dataset in which reasonably well separated sample clusters exist (in $\mathbb{R}^n$), then t-SNE will create a projection in which these clusters also appear well separated. t-SNE has few hyparameters, from which *perplexity* is the most important one, as discussed next.

In the t-SNE algorithm, data similarity in $\mathbb{R}^n$ is computed for each pair of data points $\mathbf{x}_i$ and $\mathbf{x}_j$. In detail, the method computes the probability that $\mathbf{x}_i$ picks $\mathbf{x}_j$ as its neighbor, sampled from a Gaussian distribution centered at $\mathbf{x}_i$ and of standard deviation σ_i as

$$p_{j|i} = \frac{\exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2 / 2\sigma_i^2\right)},$$

where $p_{i|i} = 0$. The parameter σ_i is computed based on *perplexity* π which is a hyperparameter of the method defined by the user. First, $p_{j|i}$ is computed for a predefined value σ_i. The next step consists in finding σ values that satisfy

$$\pi = 2^{\sum_j p_{j|i} \log_2 p_{j|i}}. \tag{2.19}$$

The σ_i values are found by solving Eqn. 2.19 by numerical methods, *e.g.*, using bisection.

After this step, a symmetric version of the similarities is computed as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}.$$

Similarity for the projected data $P(\mathbf{x}_i)$ is separately modeled by a Student's t-distribution

$$q_{ij} = \frac{\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|\mathbf{y}_k - \mathbf{y}_l\|^2\right)^{-1}}, \tag{2.20}$$

As for the similarity of the high-dimensional points, $q_{ii} = 0$ and $q_{ij} = q_{ji}$.

Having now defined similarities of points in the high-dimensional space and in the 2D projection, the aim is to construct a mapping that minimizes differences between them. This difference is measured by computing the Kullback-Leibler divergence between the respective distributions of P and Q, *i.e.*

$$KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{2.21}$$

To minimize this difference, t-SNE starts from a random initial state, or random projection $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_n\}$, and next iteratively updates the projected points $\mathbf{y}_i$ by moving them downstream in the gradient of the cost function (Eqn. 2.21).

In contrast to other projection techniques, t-SNE is strictly used for data *visualization* rather than dimensionality reduction from n dimensions to some other number of $m \ll n$ dimensions. The reason for this is that the similarity model used for the projected points (Eqn. 2.20) is specifically tailored to two (maximally three) dimensions, so as to repel dissimilar data points while grouping similar ones in the projection.

As mentioned, t-SNE's strongest point is the ability to depict the presence of well-defined clusters of similar samples, when such clusters exist in the high-dimensional data. However, an important drawback of the method is the difficulty to find a good value for the perplexity π, which can strongly depend on the input dataset. Also, t-SNE is relatively computationally expensive. Still, we weigh t-SNE's advantages as being larger than its limitations, and therefore consider it in our work in visualizing decision boundaries.

2.2.2.3 *UMAP: Uniform Manifold Approximation and Projection*

Uniform Manifold Approximation and Projection (UMAP) [87] is a dimensionality reduction technique based on the theoretical framework of algebraic topology. In contrast to t-SNE, presented in the previous section, UMAP is suitable for general dimensionality reduction. Also, UMAP can be used to create projections with *out-of-sample* capability. That is, UMAP can learn a *mapping* from $\mathbb{R}^n$ to $\mathbb{R}^2$ (rather than a *projection* of a single dataset $\mathcal{X}$ to $P(\mathcal{X})$), which can then be used to project multiple datasets, or multiple versions of the same dataset.

While discussing the theoretical foundations of UMAP is outside the scope of this thesis, we present next a brief description of the algorithm. Similarly to t-SNE, UMAP induces a representation of the high dimensional input dataset $\mathcal{X}$ based on a similarity measure $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$.

The algorithm starts by computing for each data point $\mathbf{x}_i \in \mathcal{X}$ its k nearest neighbors $NN_k^i = \{\mathbf{x}_{i_1}, \dots, \mathbf{x}_{i_k}\}$ with respect to the similarity measure d, where is k a hyperparameter of the algorithm.

For each $\mathbf{x}_i$, let

$$\rho_i = \min_{\mathbf{x}_{i_j} \in NN_k^i} \{d(\mathbf{x}_i, \mathbf{x}_{i_j})\},$$

be the minimal distance d between $\mathbf{x}_i$ and its nearest neighbors.

Similar to t-SNE, UMAP aims to numerically find the value σ_i that satisfies

$$\sum_{j=1}^{k} \exp\left(\frac{-\max\left(0, d\left(\mathbf{x}_i, \mathbf{x}_{i_j}\right) - \rho_i\right)}{\sigma_i}\right) = \log_2(k).$$

Using ρ_i and σ_i, a weighted directed graph $\bar{G} = (V, E, w)$ is constructed. The vertices are the samples from the input dataset $\mathcal{X}$. Edges are created from each $\mathbf{x}_i$ to its neighbors in NN_k^i, with edge weights computed as

$$w(\mathbf{x}_i, \mathbf{x}_{i_j}) = \exp\left(\frac{-\max(0, d(\mathbf{x}_i, \mathbf{x}_{i_j}) - \rho_i)}{\sigma_i}\right).$$

Note that $\bar{G}$ is a directed graph. To symmetrize the problem, an undirected graph G is computed using the adjacency matrix A of $\bar{G}$. We refer to the adjacency matrix of G as B, which is computed as

$$B = A + A^T - A \circ A^T, \tag{2.22}$$

where $M \circ N$ denotes the pointwise product between matrices M and N. The projected points $\mathbf{y}_i$ are now computed as the layout, *i.e.* vertex positions, of the graph G, represented by the matrix B in Eqn. 2.22. In detail, $\mathbf{y}_i$ are initialized randomly and next iteratively updated according to attraction and repulsion rules.

Similarly to t-SNE, UMAP can produce projections that exhibit clearly separated visual clusters for data points which are separated in the high-dimensional space. The method is computationally less expensive than t-SNE, has easy-to-set parameters and, as already mentioned, has the out-of-sample capability, which is important when one wants to project the same (or related) dataset(s) multiple times and compare the projections. A more detailed comparison of t-SNE and UMAP is given in [37, 87]. Given all above, we also consider UMAP, along with LAMP and t-SNE, in our work next.

2.2.3 *Inverse Projection Techniques*

In Sec. 2.2.2, we presented the overall idea of projections as functions which associate to every point $\mathbf{x}$ in the high-dimensional space a corresponding point $\mathbf{y} = P(\mathbf{x})$ in the low-dimensional space. Given this functional view, an interesting (and natural) question comes: Can we define, and compute, an inverse function P^{-1} that, given a point $\mathbf{y}$ in the low dimensional space, returns the point $\mathbf{x}$ that would have projected to $\mathbf{y}$ by using the mapping P? Or, putting it simpler: What would be the table row, in Fig. 2.7, that would correspond to *any* 2D point selected in the right image?

Before explaining why inverse projections are useful, it is important to understand that, in the above, we cannot speak about an inverse function, in the strict mathematical sense of the word, for several reasons. First, certain projection techniques do not propose a mapping from the $\mathbb{R}^n$ to the low-dimensional $\mathbb{R}^m$ *space*, but a mapping from the high-dimensional *dataset* $\mathcal{X}$ to the low-dimensional scatterplot (thus, also *dataset*) $P(\mathcal{X})$. When $\mathcal{X}$ changes, the mapping P changes too. Stronger, even when the same projection algorithm is run several times on a given dataset $\mathcal{X}$, non-parametric algorithms can generate different scatterplots $P(\mathcal{X})$, subject to various stochastic initializations. Examples hereof are LAMP and t-SNE, discussed earlier. Hence, if we aim to compute an inverse by considering *any* point $\mathbf{y} \in \mathbb{R}^m$, we actually need a mapping between the *spaces* $\mathbb{R}^m$ and $\mathbb{R}^n$. Graphically put: If we select to inversely-project a point $\mathbf{y}$ that corresponds to empty space in the scatterplot in Fig. 2.7, there is no actual data sample that P projected there. Hence, P^{-1} will have to somehow *interpolate* between the actual data samples in $\mathcal{D}$. Secondly, projection techniques do not need to be *injective* mappings: It is very well possible that, given two points $\mathbf{x}_1 \in \mathcal{X}, \mathbf{x}_2 \in \mathcal{X}$, $\mathbf{x}_1 \neq \mathbf{x}_2$, these get projected in the same location, *i.e.*, $P(\mathbf{x}_1) = P(\mathbf{x}_2)$. Principal Component Analysis (PCA) [67] is a simple example of an algorithm that can generate such issues. Hence, when considering $P(\mathbf{x}_1)$, what should be its inverse?

Regardless of the fact that we cannot formally define P^{-1} as an inverse function of P, the inverse idea can be defined in a weak sense. That is, given a projection P of a dataset $\mathcal{D}$, an inverse projection should, ideally

- associate, for every $\mathbf{y} \in P(\mathcal{X})$, the point $\mathbf{x} \in \mathcal{X}$ that projected there (if such a single point projected at that place) or, more loosely, a suitable blending (interpolation) of all points in $\mathcal{X}$ that project at, or close to, $\mathbf{y}$;
- behave in a continuous, interpolating, fashion. That is, a point $\mathbf{y}' \in \mathbb{R}^m$ which is close to several points $\mathbf{y} \in P(\mathcal{X})$ should inversely project close to the points $\mathbf{x} \in \mathcal{X}$, where $\mathbf{y} = P(\mathbf{x})$.

Within the above weak definition of inverse projections, such techniques have a number of uses. For example, assume a projection where each point visually depicts a high-dimensional data instance, such as a shape. Clicking somewhere close to a set of existing shapes, *e.g.* in the middle of their respective scatterplot points, would generate, by inverse projection, a shape that suitably blends to the existing shapes projected there. This type of technique can be very effective for generating additional data from a given, finite-size, dataset [124]. Similarly, imagine that the projected data points represent presets of the parameters of some simulation or process. Clicking somewhere between a set of such presets would generate a parameter-set that suitably blends between the respective presets [96, 147]. By extrapolation, we find inverse projections a very interesting mechanism to study in the context of exploring the decision spaces of classifiers, if these spaces are presented by means of projection scatterplots.

In contrast to (direct) projections, only a few inverse projection algorithms exist in the literature. We present the two such algorithms we are aware of next. In Chapter 6, we will propose a new method for inverse projections and compare it with these two algorithms presented below.

2.2.3.1 *Inverse LAMP*

Inverse Local Affine Multidimensional Projection (iLAMP) is an inverse projection technique based on the same theoretical foundations of LAMP, discussed in Sec. 2.2.2.1. As with LAMP, for each new point $\mathbf{y} \in \mathbb{R}^2$ to be inverted (or inversely projected), a new affine transformation has to be induced, based on a subset of the projected data points from the entire projection $\mathcal{Y}$ and their high dimensional counterparts in $\mathcal{X}$.

The iLAMP algorithm starts by finding the set $\mathcal{Y}_S = \{\mathbf{y}_1, \dots, \mathbf{y}_k\}$ of the k closest points to $\mathbf{y}$ in the projection, *i.e.* $\mathcal{Y}_S \subset \mathcal{Y}$, and their *nD* counterparts $\mathcal{X}_S = \{\mathbf{x}_1, \dots, \mathbf{x}_k\} \subset \mathcal{X}$. The goal of the algorithm is to find an affine transformation of the form $f_\mathbf{y}(\mathbf{p}) = \mathbf{p}M + \mathbf{t}$, where the matrix M and the vector $\mathbf{t}$ depend on $\mathbf{y}$, and are such that they obey the following condition

$$
\begin{aligned}
&\underset{M,\mathbf{t}}{\text{minimize}} && \sum_{i=1}^{k} \alpha_i \left\| f_\mathbf{y}(\mathbf{y}_i) - \mathbf{x}_i) \right\|^2 \\
&\text{subject to} && M^T M = I \\
&\text{with} && \alpha_i = \frac{1}{\left\| \mathbf{y}_i^S - \mathbf{y} \right\|^2}.
\end{aligned}
$$

The above optimization problem is equivalent to the minimization below

$$
\begin{aligned}
\underset{M}{\text{minimize}} \quad & \sum_{i=1}^{k} \alpha_i \, \|\hat{\mathbf{y}}_i M - \hat{\mathbf{x}}_i\|^2 \\
\text{subject to} \quad & M^T M = I \\
\text{with} \quad & \hat{\mathbf{x}}_i = \mathbf{x}_i - \frac{\sum_{i=1}^{k} \alpha_i \mathbf{x}_i}{\sum_{i=1}^{k} \alpha_i}, \\
& \hat{\mathbf{y}}_i = \mathbf{y}_i - \frac{\sum_{i=1}^{k} \alpha_i \mathbf{y}_i}{\sum_{i=1}^{k} \alpha_i}.
\end{aligned}
$$

Given M, found by solving the optimization problem above, the inverse projection of the $2D$ point $\mathbf{y}$ is given by

$$
P^{-1}(\mathbf{y}) = \left(\mathbf{y} - \frac{\sum_{i=1}^{k} \alpha_i \mathbf{y}_i}{\sum_{i=1}^{k} \alpha_i} \right) M + \frac{\sum_{i=1}^{k} \alpha_i \mathbf{x}_i}{\sum_{i=1}^{k} \alpha_i}.
$$

As with LAMP, the algorithm to find M relies on SVD, which has fast and readily available implementations in many numerical methods packages.

2.2.3.2 *RBF based Inverse*

As an alternative to iLAMP, Amorim et al.[5] use *radial basis functions* (RBFs) to interpolate among $2D$ and nD data points to output a high-dimensional candidate for a given point in the 2D projection plane. RBFs are real valued functions that depend only on the distance of the argument to a given point, *i.e.*, satisfy the property $\varphi(\mathbf{x}) = \varphi(\|\mathbf{x} - \mathbf{c}\|)$, where $\mathbf{c}$ is a reference point and $\|\cdot\|$ is a distance metric.

While iLAMP performs data interpolation weighted by Euclidean distance, RBF-based inverse projection uses *kernels* to estimate data similarity. Kernels commonly used in practice are the Gaussian $\varphi(r) = e^{-\epsilon r^2}$ and Multiquadrics $\varphi(r) = \sqrt{1 - (\epsilon r)^2}$, where ϵ is a parameter controlling the kernel's shape. Let $P^{-1}(\mathbf{y}) = (P_1^{-1}(\mathbf{y}), \ldots, P_n^{-1}(\mathbf{y}))$ be the nD inverse of the 2D point $\mathbf{y}$. The k^{th} coordinate $(1 \leq k \leq n)$ of the inverse of a given $2D$ point $\mathbf{y}$ is obtained by interpolation as follows

$$
P_k^{-1}(\mathbf{y}) = \sum_{i=1}^{N} \lambda_i^k \varphi(\|\mathbf{y}_i - \mathbf{y}\|) \tag{2.23}
$$

The coefficients λ_i^k from Eqn. 2.23 are determined from the constraint that the inverse mapping suits the data for which we know actual projection locations, *i.e.*, given by $P(\mathcal{X}) = \mathcal{Y}$. That is, for a $2D$ point $\mathbf{y}_j \in \mathcal{Y}$ and its high dimensional counterpart $\mathbf{x}_j \in \mathcal{X}$, RBF-based inverse method finds λ that satisfy

$$P_k^{-1}(\mathbf{y}_j) = \sum_{i=1}^{N} \lambda_i^k \varphi(\|\mathbf{y}_i - \mathbf{y}_j\|) = \mathbf{x}_{jk} \tag{2.24}$$

The coefficients λ_i^k for a given coordinate k can thus be found by solving the linear system of equations below

$$\begin{bmatrix} \varphi_{1,1} & \varphi_{1,2} & \cdots & \varphi_{1,N} \\ \varphi_{2,1} & \varphi_{2,2} & \cdots & \varphi_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{N,1} & \varphi_{N,2} & \cdots & \varphi_{N,N} \end{bmatrix} \begin{bmatrix} \lambda_1^k \\ \lambda_2^k \\ \vdots \\ \lambda_N^k \end{bmatrix} = \begin{bmatrix} x_1^k \\ x_2^k \\ \vdots \\ x_N^k \end{bmatrix} \tag{2.25}$$

where $\varphi_{i,j}$ is the value of the RBF φ for reference point i and evaluation point j and x_i^j is the j^{th} dimension of point i of $\mathcal{X}$. Thus, to find all coefficients λ needed to evaluate the interpolation in Eqn. 2.23, it is necessary to solve n systems of equation, where n is dimensionality of $\mathcal{X}$.

This method can achieve smoother and more natural-looking inverse projections than iLAMP, as demonstrated by several use-cases [5]. However, its computational costs are also higher than iLAMP. Given this trade-off, which is, we argue, application dependent, we will use both iLAMP and the RBF method to invert projections in our work further on.

2.2.4 *Visual analytics techniques for classifier engineering*

Even before the advent of what is currently known as Explainable Artificial Intelligence (XAI) [3], numerous techniques have been proposed at the crossroads of Machine Learning, Artificial Intelligence, data science, and visualization for helping the *engineering* – that is, selection, construction, fine-tuning, and validation – of classifiers. We discuss below a number of salient techniques in this family. As with projections (Sec. 2.2.2), the complete set of such techniques is too wide to summarize here. As such, we focus below on methods which are either broadly accepted and used (thus, with which our own proposals will compete), or methods which are technically related to our proposals. We organize the surveyed techniques into three groups, depending on the kind of *information* they focus on (classes, observations, or the classifier's architecture), and by analogy with how we organized the more general visualization techniques for high-dimensional data (Sec. 2.2.1), as follows.

2.2.4.1 *Class-centric techniques*

Class-centric techniques focus on understanding how a classifier behaves in relation to the *classes* it is supposed to infer from data. That is, it allows one to reason about aspects such as the general classification accuracy (how well the classifier infers correct labels for all the present classes), per-class accuracy (how well the classifier infers correct labels for a specific class), and how these accuracies depend on various parameters. Techniques in this family include measuring aggregated metrics for an entire dataset (or class in the dataset), such as precision, recall, sensitivity, F1 score, and specificity [153]. These can be next presented by means of simple graphical metaphors such as truth tables and confusion matrices. At a more refined level, Receiver Operating Characteristic (ROC) plots [45] can be created to show how sensitivity and specificity relate to each other for a whole range of model hyperparameters, therefore allowing users to make informed trade-offs between the two parameters.

Class-centric techniques are the earliest, and still most used, exploration techniques for classifiers, for a number of good reasons. Following the set of requirements we outlined in Sec. 1.4, these techniques are definitely simple to use and interpret, generic (work for any classifier), and computationally very scalable. Their visual presentation is also very compact and uses only simple means such as tables, charts, and function plots, which users are expected to understand easily. However, they also have disadvantages: They only tell *aggregated* insights at class, or higher, level. We can see for instance what a certain accuracy is for an entire test set, but we cannot see which *parts* (in terms of sample subsets) of the respective test set are more (or less) prone to classification problems. By implication, this means that these techniques are good for telling us how well a classifier works, but in the case its performance is unsatisfactory, they do not generally help one with engineering the classifier to improve its performance. More specifically to our research goals, they do not tell anything about where the decision boundaries exist in the data space and/or how these are created from the training samples.

2.2.4.2 *Observation-centric techniques*

Observation-centric techniques recognize the above-mentioned limitations of class-centric techniques and proceed differently towards explanation. Rather than selecting classes, they select (groups of) instances $\mathbf{x}_i$ or feature vectors $\mathcal{X}^j$ as the element to base explanations on. The simplest, and widely used, such observation technique is the scatterplot, constructed by a projection, of samples, colored by class or misclassification information discussed in Sec. 2.2.2). As outlined there, such scatterplots already can show which samples are prone to misclassification, either in general, or specific to a given class.

While such scatterplots are simple and efficient to compute, they have a major drawback: Depending on the size of S (sample count), how it is distributed over the space, and how well the projection P preserves distances or neighborhoods, gaps will appear between the points of the resulting 2D scatterplot. One can only *guess* what happens between such samples.

To mitigate this general issue of scatterplots, so-called *image-based* methods, or *dense maps*, have been proposed. The key idea is to color every pixel $\mathbf{p_i} \in \mathbb{R}^2$ of the target image to represent information pertaining to that pixel in the data space $\mathbb{R}^n$. There are several ways to "fill in" these pixels and thereby create a dense, compact, image from the sparse, discrete, scatterplots, as follows.

Prior to applications in ML, several researchers have recognized that the discrete nature of scatterplots poses interpretation problems, especially when the underlying data is drawn from a continuous phenomenon. Early techniques converted scatterplots S (not necessarily coming from a projection) to continuous scalar fields $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^+$ using so-called kernel density estimation (KDE) methods [134]. This can be done by simply convolving points $\mathbf{x} \in S$ by a suitably chosen (typically, Gaussian or Epanechnikov) 2D isotropic kernel. The kernel radius acts as a low-pass filter: Small values yield relatively discrete images, showing concentrated peaks on the image ϕ where many samples in S are close to each other. Larger values "blur" the KDE, by showing coarse-scale groups and structures. The produced density map ϕ can be next visualized by mapping it to color, brightness, or opacity, leading to the well-known "heatmaps" in visualization. This way, one can easily spot high-density regions of samples in a scatterplot at a user-chosen scale, given by the kernel radius parameter. Early examples of such techniques include graph splatting [78] for visualizing graph drawings.

Closer to our machine learning context, Martins *et al.* propose several dense maps to encode the per-pixel errors created by dimensionality reduction methods [85, 86]. Similar dense map methods are used to encode the (categorical) identity of dimensions that make close points in a projection similar to each other [131]. Variants of this idea have been proposed to handle categorical data, using a Voronoi cell sampling of the image space, rather than a uniform pixel grid [8, 19]. Key advantages of image-based methods are their ability to use every available pixel to show information, which increases the chance that complex data patterns are spotted without the need for the user to "guess" what happens between discrete samples; the lack of occlusion present in discrete methods, such as scatterplots; and the ability to handle large datasets by aggregating data over the available pixels. Given the above, we deem that image-based techniques are ideally suited for our goal of visualizing classifier decision zones and boundaries.

However, while such methods can handle arbitrary high-dimensional datasets, none of them was adapted to show classifier decision boundaries, with one notable exception: The well-known TensorFlow toolkit [1] contains a simple application which creates a dense map where pixel $\mathbf{p_i}$ is color-coded to indicate the class label, and corresponding weight, that a neural network achieves for a sample $\mathbf{x_i}$ that would correspond to $\mathbf{p_i}$. Figure 1.2 discussed in Chapter 1 shows an example of this visualization. However, this works only because the input space (for the toy examples used in [138]) is two-dimensional, so the $\mathbb{R}^n$ to $\mathbb{R}^2$ mapping is trivial *and* invertible, being the identity function.

Closer to our scope, Hamel has proposed dense self-organizing image-based maps to visualize classifier decisions for high-dimensional feature spaces [53]. However, this method only handles Support Vector Machine (SVM) classifiers. More importantly, the actual decision boundaries, while plotted on the respective maps, seem to be manually constructed by the user rather than by the method. Migut *et al.* actually construct and visualize such decision boundaries for high-dimensional data classification [88]. However, they use for this a sequence of 2D projections, each along a pair of dimensions; hence, the user has to mentally infer the actual position of the high-dimensional boundaries by interactively correlating multiple projections.

Last but not least, the LIME technique [118] proposes an ambitious set of techniques for “explaining the predictions of *any* classifier”. Briefly put, given such a classifier, which has a globally complex, and generally unknown, decision boundary, LIME locally approximates this boundary by densely sampling the prediction function and next fitting a linear boundary to the obtained labels. This way, while it is still not possible to get insights in the global decision boundaries, good (linear) local approximations can be computed on demand. In terms of visualization, LIME uses various standard instruments, such as tables and charts highlighting the types of observations that are most relevant for a given classification decision. Alternatively, when the feature space is easy to represent visually, such as in the case of images processed by deep learning (where each pixel is basically a feature), so-called activation maps can be computed, which highlight pixels in a given input sample (image) that have been most responsible for its classification in a certain category. Overall, LIME is a very powerful technique for analyzing the behavior of a classifier. However, it does not explicitly bring new insights into how the respective decision boundaries look like.

2.2.4.3 *Architecture-centric techniques*

A third and last class of explanatory techniques focuses on the *architecture*, or internals, of a classifier. Such techniques are especially useful in case of classifiers that have (very) complex architectures, such as neural networks. These consist of thousands up to hundreds of thousands of units (neurons) connected by up to millions of weights. As such, their overall operation is largely still a black box. Explanatory techniques in this class aim to shed light upon the roles of various layers, units, and connections during both training and inference, and therefore both understand how and what the network has learned, and how to improve this process. A recent survey of techniques in this class is given in [49].

The area of architecture-centric techniques is growing very fast, with tens of such techniques launched in recent years, fed by the growing interest in deep learning. However, from the perspective of the research in this thesis, such techniques occupy a peripheral roles, for two reasons. First, they are specific to neural networks (or any other given architecture) only, whereas we aim at generic explanatory techniques. Secondly, these techniques do not explicitly aim to visualize the place, shape, and nature of decision zones in the data space, but rather concentrate on understanding the intermediate (latent) representations that are generated by a given classifier.

2.2.5 *Conclusions*

In this chapter, we have reviewed related work to the areas at the crossroads of which our research question lies, namely machine learning (with a focus on classification) and visualization and visual analytics (with a focus on assisting classifier engineering). Focusing on the second of the above topics – which is related directly to our aim – we conclude that multidimensional projections are a better candidate than other high-dimensional data visualization techniques for visually exploring the type of high-dimensional datasets encountered in classifier engineering. Within this area, we have identified three projection techniques and two inverse projection techniques which present good properties in terms of results' quality, computational scalability, genericity, and ease of use. Separately, we have seen that image-based visualization methods match well, on the one hand, with the type of data we encounter in classification problems, and on the other hand with the visual metaphor used by projections. Hence, these methods are interesting candidates to study further when addressing our research question.

In the same time, we have seen that the problem of visually depicting (and further exploring) the decision boundaries and decision zones of any classification models is barely touched in the literature. In the remainder of this thesis, we will therefore focus on the above-mentioned instruments – direct and inverse projections and image-based visualization methods – and show how we can combine, adapt, and extend, such methods to address our goals.

DEEP FEATURE EXTRACTION EVALUATION

3

3.1 INTRODUCTION

As explained in Chapter 2, deep learning methods are among the state of the art techniques for constructing classification models for complex data. However, as also explained there, engineering such models is, in general, far from trivial, and raises questions related to the suitable selection of training sets, network architecture, and hyperparameter values. In the following chapters, we will introduce several visual analytics techniques that aim to help this process, specifically by depicting decision zones and boundaries and showing how these can provide feedback to the model engineer. In this chapter, we aim to justify the need for such assisting techniques by an actual example of classifier engineering. For this, we pick a real-world image classification problem, and show the steps required for the construction of a non-trivial end-to-end classification model for that problem. By doing this, we outline the types of questions and challenges that model engineers are typically confronted with in their work, and also outline the limitations of classical support tools available in this process – thereby indirectly motivating our claim for the need for better tools.

3.2 PROBLEM CONTEXT

Deep Learning methods are the current state of the art tool to perform natural image classification, as well as other pattern recognition tasks such as speech recognition and shape analysis, among others. The influential work of Krizhevsky et al. [73] has demonstrated how Graphics Processing Units (GPUs) could make the training of complex Deep Convolutional Neural Networks feasible in acceptable time frame. More importantly, they achieved breakthrough results on ImageNet Challenge [123], results largely impacting computer vision and neural networks by stimulating novel research in the area. Since the proposal of AlexNet [73], a number of other Deep Neural Networks were developed

---

This chapter is based on the following publication:
F. C. M. Rodrigues., N. S. T. Hirata., A. A. Abello., L. T. de la Cruz., R. M. Lopes., and R. H. Jr.. Evaluation of transfer learning scenarios in plankton image classification. In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP,*, pages 359–366. SciTePress, 2018

and achieved ground-breaking results on different image classification challenges [58, 135, 144].

To properly train Deep Neural Networks, a large amount of data is necessary, besides extensive parameter-tuning, since such models often consist of millions of parameters. To avoid overfitting, diverse and rich datasets are often mandatory. However, in many real-world scenarios, labeled data is costly and scarce. One example of such scenario is the study of plankton populations, an important research area as they form the basis of aquatic food webs and exert a major influence on material cycles relevant to global climate change, such as carbon dioxide and methane. To precisely estimate the spatial and temporal distributions of planktonic organisms in the ocean, challenging image acquisition efforts are necessary. In order to employ obtained images to train Machine Learning models, the time-expensive task of manual image labeling is necessary, which requires trained domain-expert individuals.

In ML, an usual approach to deal with the above mentioned problem of training data unavailability is Transfer Learning (TL) [98]. TL expresses the concept of using or adapting a model induced in a specific context to another context. For instance, using or adapting a model induced using a plankton dataset from Atlantic ocean to classify another one from the Pacific ocean.

In this chapter, we present how to employ the simplest form of transfer learning based on Deep Neural Networks to solve a real problem of plankton classification. By leveraging on the public availability of labeled planktonic data, DNNs can be used to help the training of classifiers on local *in-house* collected data, that are usually small. We note that plankton image classification using CNNs started to be considered only recently [4, 28, 112] and, in particular, transfer learning of features computed by CNNs [93] has not been explored much yet in this context. Hence, we also aim at deepen our understanding of transfer learning, in special for planktonic data.

The remainder of this chapter is organized as follows. Section 3.3 presents a more detailed description of TL using DNNs. Section 3.4 explains how different TL scenarios were setup to train and compare different models and datasets. Section 3.5 gives a brief explanation of the source and target datasets used as well as the DNN models employed in the following experiments. Section 3.6 presents the results obtained for each dataset/network combination. Finally, Section 3.7 presents how one could extract useful insights from this experiment and points to limitations of this approach, motivating the need for better visual analytics tools to perform classifier engineering and understanding.

3.3 DEEP FEATURE EXTRACTION

Representations learned by CNNs are reported to be very useful for the classification of data, even in distinct domains [14, 165]. The usual approach to exploit this is to select an intermediate layer as a target layer, freeze it and its preceding layers and adjust the subsequent layers. The earlier the layer chosen, the more general and therefore, more transferable the representation is [165], but also the more data is necessary to adjust it, since it has a higher dimension. The adjustment of subsequent layers may be done via fine-tuning, continuing training with new samples, or by training an entirely new classifier from scratch using the output of the intermediate target layer as features, which is called (deep) feature extraction. In this work we chose the latter option, using pre-trained CNNs as feature extractors.

3.4 EXPERIMENT SETUP

Plankton communities form the basis of aquatic food webs and exert a major influence on material cycles relevant to global climate change, such as carbon dioxide and methane. Therefore, it is essential to understand the spatial distribution and temporal variability of planktonic organisms in the ocean.

Deep feature extraction based approaches to data classification enables the easy application of deep learning techniques to solve different problems, specially when the process of labeled data acquisition is costly, as is the case for many important "local" problems.

Planktonic image classification is an example of such problem. Understanding the spatial distribution and temporal variability of planktonic organisms in the ocean is essential to the study of different important topics, such as global climate change.

Thus, to classify our in-house dataset, which we refer to as LAPSDS, we resort to adapt pre-trained models as feature extractors by taking advantage of the fact that there exists a public available massive dataset of plankton images used in Kaggle's National DataScience Bowl (NDSB) competition, via the In Situ Icthyoplankton Imaging System (ISIIS)[1], and deep neural networks trained on it.

Although there are differences in the datasets with respect to the classes of plankton species they include, either because a particular species or class in one of the datasets is not in the other or because some artificial classes are created based on other criteria not related to taxonomy, it is reasonable to expect that they could be efficiently classified by similar sets of features. To further investigate the quality of the features obtained from this process, we also employ a different CNN trained on this same dataset and on ImageNet [123], which contains images from a completely distinct domain. By using CNNs and external domain source datasets, we would like to understand how transfer learning performs and whether an external dataset will help or not the classification of our data.

---

[1] `https://www.kaggle.com/c/datasciencebowl`

The choice of the datasets is justified by the fact that a relatively mature stage of CNN development has been already achieved for both domains. In addition, the first dataset is of a similar domain to ours, while the second is of a completely distinct domain (several images collected from the Internet).

The experiments have been designed to answer the following questions:

- how DeepSea trained on ISIIS (NDSB competition) – DeepSea(ISIIS) – will perform on our in-house dataset (LAPSDS)?
- how classifiers using features extracted from DeepSea(ISIIS) will perform on LAPSDS?
- how classifiers using features extracted from AlexNet trained on ISIIS – AlexNet(ISIIS) – will perform on LAPSDS?
- how classifiers using features extracted from AlexNet trained on ImageNet – AlexNet(ImageNet) – will perform on LAPSDS?

In addition to these TL scenarios, we also consider the traditional feature extraction approach that will serve as a baseline. Diagram in Fig. 3.1 summarizes the scenarios to be evaluated. Four sets of features are extracted from LAPSDS and they are used to train SVM classifiers as detailed ahead in Section 3.4.

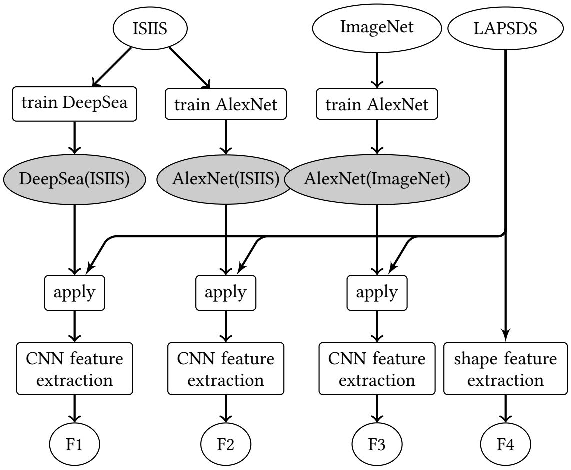


Figure 3.1: Deep feature extraction scenarios considered here. ISIIS ImageNet and LAPSDS denote image datasets, gray shaded nodes indicate the pre-trained CNNs, and CNN feature extraction consists of extracting the values from a specific layer of a CNN, after a forward pass of samples in LAPSDS.

3.5 DATASETS AND NETWORKS

In situ plankton images have been acquired with a submersible instrument developed at our lab LAPS-IOUSP[2]. The instrument has been vertically deployed between surface and 30m depth off the lab base[3] and gray-scale images were acquired at approximately 15 frames per second, with dimensions of 2448 × 2050 pixels and resolution of $\sim 5 \mu m$. Image stacks belonging to the same vertical profile were converted into video files to mitigate data storage and management. A total of 230,000 Regions of Interest (ROI) were extracted from 16 selected videos and 5175 ROIs were used in the creation of in-house dataset. A labeling process was carried by plankton experts belonging to the same lab.

LAPSDS is composed of 20 classes containing at least 100 samples each, and as expected, the number of images varies from class to class. Table 1 shows the class distribution of the dataset, as well as the name and the identifier number of each class. Instances of some of the classes are shown in Fig. 3.2.

ID	H classes	Size	ID	H classes	Size
0	appendicularia_shape_s	216	10	detritus_uf_stick_bw	286
1	appendicularia _curve	114	11	dinoflagellates_tripus_2	242
2	cladocera	435	12	dinoflagellates_tripus	316
3	copepod_calanoid	315	13	nauplii	465
4	copepod_cyclopoida	106	14	phytoplankton_0	259
5	copepod_poecilostomatoida	163	15	phytoplankton_1	127
6	detritus_df_bk	288	16	phytoplankton_5	159
7	detritus_uf_dot_bk	344	17	chaetocero	546
8	detritus_uf_dot_bw	274	18	diatoms_coscinodiscus	120
9	detritus_uf_stick_bk	152	19	shadow	249

Table 1: Histogram of classes of the LAPSDS.

In-situ images are prone to natural variability in illumination, turbulent flow and turbidity, among other factors, which may compromise image quality because ROIs from different videos may have different background intensities (see Fig. 3.2). Thus, for convenience, the background of the ROIs have been removed using a technique of background subtraction adapted to deal with illumination changes [65]. An example is shown in Fig. 3.3.

---

[2] Laboratory of Plankton Systems, Oceanographic Institute, University of São Paulo, Brazil

[3] (lat:-23.499913, long:-45.119381)

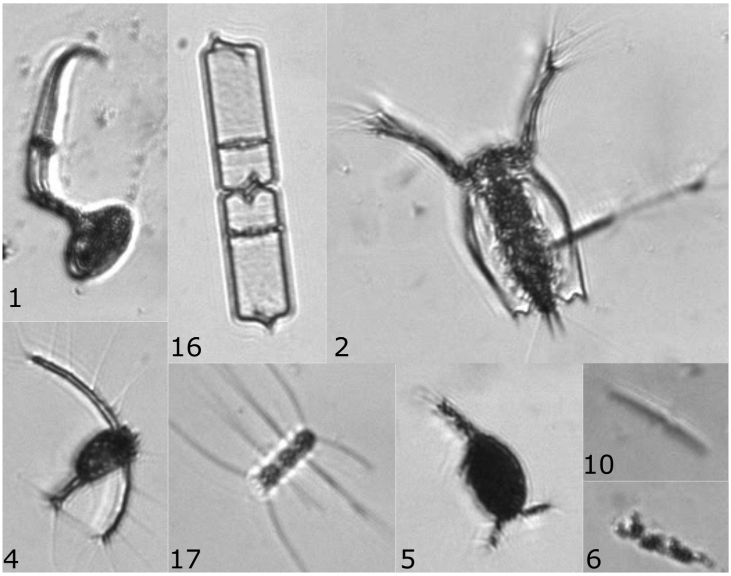


Figure 3.2: Image sample from LAPSDS. Number on the ROI indicates the class that they belong to.

3.5.0.1 *Kaggle's National Data Science Bowl*

The National Data Science Bowl (NDSB) was a competition hosted by Kaggle in a collaboration with Oregon State University's Hatfield Marine Science Center. Several research teams competed to develop and train supervised classifiers, given a dataset provided by the Hatfield Marine Science Center [26].

According to the competition organizers, the images were collected in the Straits of Florida using an underwater imaging system called ISIIS

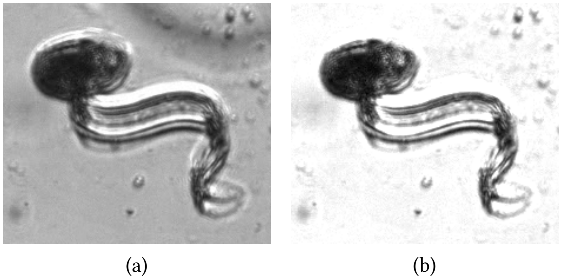

(a)
(b)

Figure 3.3: Background removal example: (a) Original image, labeled as "appendicularia_shape_s" and (b) result of the background removal of image in (a).

(In Situ Ichthyoplankton Imaging System). It captured high-resolution continuous images that were parsed in 2048x2048 pixel frames. The resulting frames were thresholded and segmented. Finally, regions of interest were extracted and became the images that comprise the dataset after being annotated by the Marine Science Center's personnel.

The dataset was divided by taxonomy, behavior and shape into 121 classes. Each class contained between 9 and 1979 individual examples, totaling 30,336 images.

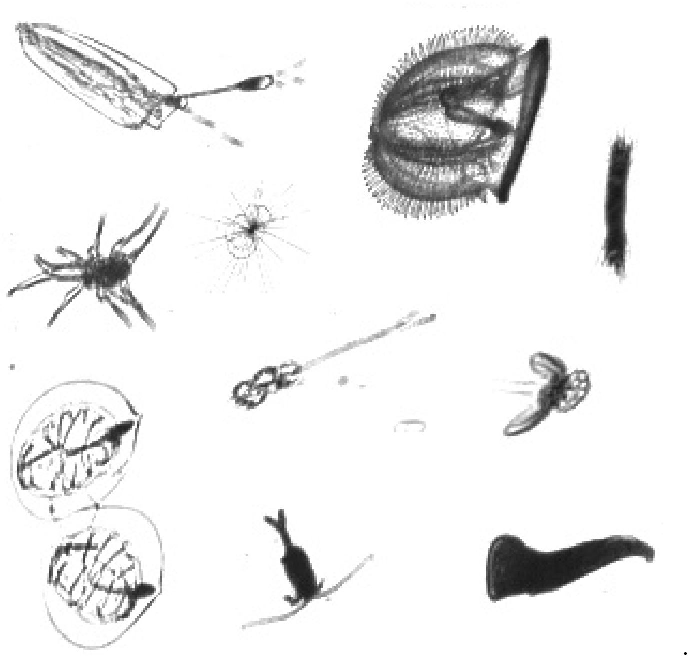


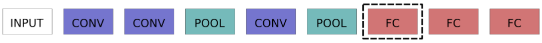
Figure 3.4: Assorted plankton from the ISIIS dataset. Each sample is from a different class. Note the absence of background.

3.5.0.2 *ImageNet*

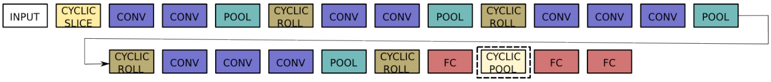
ImageNet is a dataset that became one of the benchmarks for object classification and detection. It is comprised of over 14 million images divided into 1000 classes hierarchically subdivided [123]. The classes subjects range from human persons to animals and fungi to everyday objects, constituting a very general dataset. Since 2010 a competition including diverse tasks such as classification and detection on pictures or video on this dataset is held each year.

3.5.1 *CNN MODELS*

The two network architectures used in this work are from winning teams in computer vision competitions. They are the AlexNet [73], from the 2012 ImageNet Large Scale Visual Recognition Competition (ILSVRC), and a model from the "Deep Sea" team, that won Kaggle's NDSB in 2014.



(a) AlexNet



(b) DeepSea's convroll4 network

Figure 3.5: Neural Networks architectures used in the experiments. Although DeepSea's model is much deeper than AlexNet, it has less parameters (*i.e.* filters in Convolutional layers and units in Fully Connected layers) to fit during the training. The dashed boxes indicate which layer was used in the transfer learning experiments.

3.5.1.1 *AlexNet*

AlexNet is a Convolutional Neural Network model that was introduced in the ILSVRC held in 2012. Under the team name of “SuperVision”, it won both the classification and localization tasks by a large margin[4], being the first case of success in applying this kind of model in the competition and establishing a strong trend of its use in the next years.

This model introduced and popularized a lot of novelty features for improving training time, performance and reducing overfitting including, but not limited to: ReLU nonlinearity as activation function, Dropout as means of reducing overfitting and Local Response Normalization. We refer to the original paper for a more detailed explanation of these innovations and their impact [73] (see Fig. 3.5(a) for a representing diagram of the CNN).

We did not explicitly train AlexNet model in the ImageNet dataset, but used instead a pre-trained model with available weights online [5]. In order to feed our images to this model, a couple minor modifications were required. First, ImageNet samples are RGB images, hence to feed our in-house dataset, which is composed of grayscale images, to this network we chose to repeat the same values across the three input channels. Second, since the required input is much larger than the average of ours images, we decided to resize them via a wrap padding tactic, in which

---

[4] http://image-net.org/challenges/LSVRC/2012/results

[5] https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet

the image was centered and repeated across each axis. Furthermore we also subtracted each channel with the mean of the training set of said channel, as this information was also available to us.

The AlexNet implementation that was trained on ISIIS dataset was heavily based on DeepSea's model, following exactly the same training procedure for both networks (*i.e.* data preprocessing and data augmentation). Thus, this network's input expects grayscale images with size 95x95 and its final layer contains 121 units.

3.5.1.2 *Deep Sea's Model*

Deep Sea was the winning team of the Kaggle NDSB competition. They used an ensemble of multiple deep learning models with minor differences to improve generalization. We used the most simple model available, consisting solely of a CNN, which here we call DeepSea.

The main innovation brought by the team was a couple of layers designed to increase the network robustness to cyclic variation [33]. In the "cyclic slice" layer the input is rotated four times and processed separately by the network from that point onward. Then, in the "cyclic roll" layer, the feature maps from the four paths are permuted and interchanged. Eventually, in the “cyclic pooling” layer the four network paths are merged again into a single one. We again refer to the paper on this architecture for a more detailed explanation [33] (see Fig. 3.5b for a diagram of the CNN).

3.5.2 *Feature extraction*

Deep learning models are in general trained in a *end-to-end* fashion, that is, images (or other raw data) are given as input in one end and classes are returned as output in the other end, as mentioned in Chapter 2. The last layer of a neural network classifier simply performs the classification task on its input, hence we can see the hidden layers as performing suitable data transformations that will lead to a small classification error, and the output of a given layer is a feature vector that is input to the next. Next, we detail how we collect those features for each network, *i.e.* define from which layer we collect the output, and also describe the set of shape features they will be compared to.

3.5.2.1 *Deep features*

Features From DeepSea (ISIIS): The features were extracted from the output of the last Cyclic Pooling Layer, as shown in Figure 3.5b highlighted by enclosing dashed lines, resulting in 256 features per images. These features correspond to F1 in the diagram of Fig. 3.1. In a Cyclic Pooling Layer the effect of rotations introduced by previous Cyclic Slice and Cyclic Roll layers are undone, hence capturing the output from this layer is the most appropriate choice since we can leverage on the learned invariances.

Features from AlexNet (ISIIS) and AlexNet (ImageNet): From the two pre-trained AlexNet, AlexNet(ISIIS) and AlexNet(ImageNet), features were extracted from the first fully connected layer, as shown in Figure 3.5a highlighted by enclosing dashed lines, resulting in 4096 features per image. These features correspond to F2 and F3, respectively, in the diagram in Fig. 3.1.

3.5.2.2 *Shape Features*

We extracted 74 features commonly used in traditional shape recognition procedures. They are divided into the following three categories:

- 54 shape features (area, perimeter, solidity, convexity, etc). Most of the feature descriptors are implemented in the OpenCV library and they are usually presented in automatic plankton classification works that use shape features [17].
- 10 from Local Binary Patterns (LBP) histograms [92] extracted using a 3×3 window.
- 10 from Haralick descriptors, extracted from the co-occurrence matrix [55].

Shape and LBP features are extracted from the images segmented using Otsu's threshold [95]. Haralick's descriptors are extracted from graylevel images. These features correspond to F4 in the diagram in Fig. 3.1.

3.6 CLASSIFIER EVALUATION

To train and evaluate the SVM classifiers with respect to each of the four feature sets, namely features extracted from DeepSea (ISIIS), from AlexNet trained with planktonic data (NDSB), from AlexNet trained on natural images (ImageNet), and regular, "classic", shape recognition features. We performed a 9:1 train-test split that preserved class proportions. This split resulted in a training set of 4658 and a test set of 517 samples.

Before training, a data normalization to convert all feature values to the $[0, 1]$ range was applied to each individual feature of the four feature sets, that is a simple linear scaling in which the highest value for each feature in the whole training set will be mapped to 1 and the lowest value will be mapped to 0. The normalization parameters were inferred using the training samples only in order to not add bias to the classifier. Test samples were then transformed by those same parameters.

Sklearn's [104] grid search with cross-validation was employed to explore the space of possible parameters for SVM, namely the kernel type, value of C and, if a RBF kernel was used, γ values. In this work, we considered *linear* and *RBF* kernels, $C \in \{1, 10, 100, 200\}$ and $\gamma \in \{0.01, 0.001, 0.0001, \frac{1}{nf}\}$, where nf is the number of features, a well-known heuristic for setting γ. The best parameters found for each feature set are displayed in Table 2. The same table also shows the overall accuracies computed on the test set.

Feature extractor	SVM parameters			Acc.
	kernel	C	γ	
DeepSea(ISIIS)	rbf	100	0.01	84%
AlexNet(NSDB)	rbf	10	0.01	81%
AlexNet(ImageNet)	rbf	100	0.0002	80%
Shape Features	linear	100	-	72%

Table 2: Table summarizing the results obtained from different transfer learning scenarios. The value of 0.0002 for γ was selected because of the $\frac{1}{nf}$ option. Accuracy refers to the test set.

Global accuracy alone, especially in cases such as ours, where the compared methods present similar performance, is not too informative. To better understand the results, we also plotted a *confusion matrix* (Fig. 3.6) for each feature set.

As it can be seen in Fig. 3.6, the first (leftmost) plot corresponding to DeepSea(ISIIS), which achieved the best performance, has a darker diagonal compared to the other plots. Confusion is larger in the last (rightmost) plot, which corresponds to the experiment using shape features. In general, there is confusion between class 3 (copepod_calanoid) and classes 4 (copepod_cyclopoida) and 5 (copepod_poecilostomatoida); between classes 7 (detritus_uf_dot_bk) and 8 (detritus_uf_dot_bw), and beween classes 9 (detritus_uf_skick_bk) and 10 (detritus_uf_stick_bw).

Figure 3.7(a) presents some examples of copepods subtypes that can confuse the classifiers. The figure is organized in three columns, one for each copepod subtype: column 1 shows four examples of calanoids; column 2 shows three examples of cyclopoida; and column 3 shows four examples of poecilostomatoids. Each image is labeled with zero to four colored squares that indicate which of the four considered classifiers (DeepSea, AlexNet (ISIIS), AlexNet (ImageNet), and Shape Features) could correctly classify that image. As one can see, the plankton belonging to these classes are similar in several aspects and it is not difficult to understand why these classes cause confounding errors. A similar scenario has been found for detrital particles. Figure 3.7(b) presents a similar set of images of examples of detritus subtypes (detri-

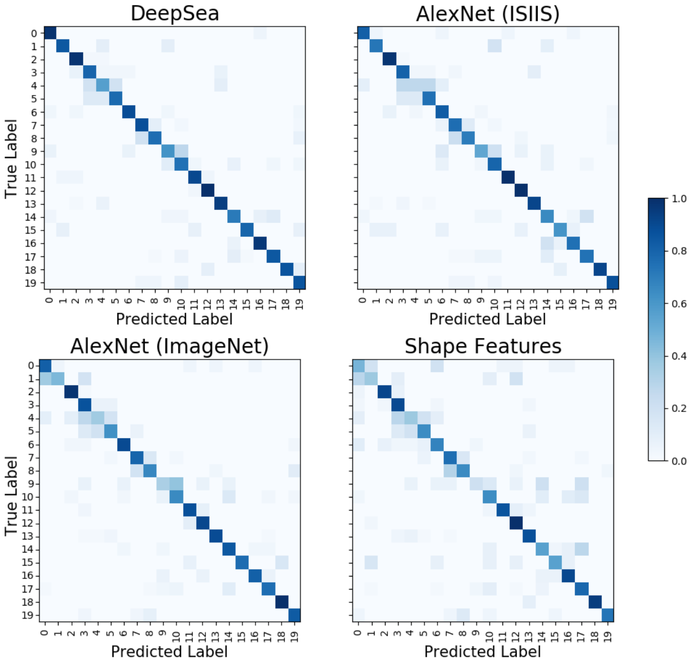
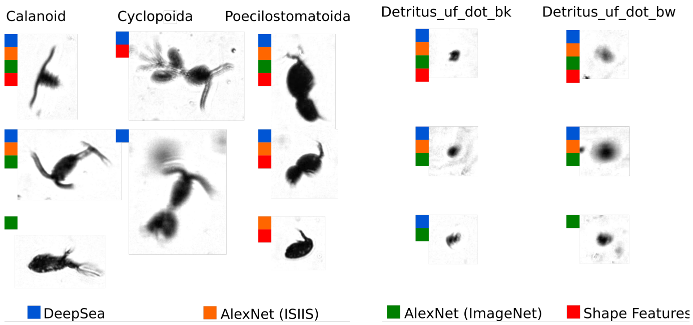


Figure 3.6: Confusion matrices for each classifier trained on different sets of features. Top row we shows the confusion matrix for the classifier trained on features from DeepSea network trained on ISIIS (left) and AlexNet also trained on ISIIS dataset. Bottom row shows matrices for AlexNet trained on ImageNet and for classic shape features.

tus_uf_dot_bk and detritus_uf_dot_bw) that can confuse several of the considered classifiers.

Figure 3.8 shows another view of the obtained results. We show here a bar chart displaying the accuracy of each classifier per class. Classes 2, 11, 12, 13, and 18 were well classified by all the four classifiers and therefore they could be considered as the "easy" classes. On the other hand, classes 4 and 9 are those where most classifiers did poorly, and thus they are the hardest ones. Classes 0 and 1 are those with the largest variation between the best and worst performing classifiers.

Hand designed features performed clearly worse than any of the CNN extracted ones. Although no careful feature selection was performed, it is also true that no careful deep feature extraction was performed. Thus, in a situation where a quick solution is required, making use of a pre-trained CNN could be more effective than using a large set of hand designed feature extractors.



(a) Cyclopoida, Calanoid, Poecilostomatoida

(b) uf_dot_bk, uf_dot_bw

Figure 3.7: Two sets of plankton images from confounding classes.

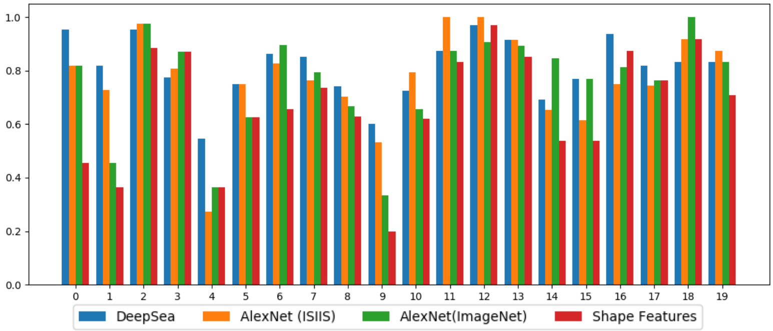


Figure 3.8: Class accuracy histogram.

3.7 DISCUSSION AND CONCLUSION

We have presented an evaluation of transfer learning scenarios in the context of plankton image classification. We have used CNNs pre-trained on external datasets as feature extractors from our in-house dataset images. In particular, we have considered two very distinct external datasets, one of plankton images (and thus similar to our data) and another of natural images (ImageNet), and the corresponding "winning" CNN architectures. Transfer learning experiments showed that the architecture developed for plankton images (DeepSea) performed better than the architecture developed for natural image classification (AlexNet), even when both were trained with the same plankton image dataset. We also observed that AlexNet trained on natural images performed almost as well as the same network trained on plankton images. These two observations indicate that, in transfer learning using CNNs, the architecture may play an important role, even larger than the dataset per se. To complement these observations, it would be interesting to train DeepSea with ImageNet and evaluate how well it will perform on our data.

As expected, features extracted from pre-trained CNNs performed better than hand crafted ones. Although the experiments in this chapter were designed to evaluate different deep feature extraction scenarios, it is clear that deep learning techniques can be promptly applied to different problems, even when dataset size is small. The available technology is mature and accessible with respect to both software and hardware, besides the public availability of data from different domains, creating a low-entry barrier environment to apply deep neural networks to different problems.

To support the analysis and comparison of the constructed classifiers, we used only simple visualization tools, which are typically encountered in many classifier engineering workflows. While providing useful insights, we also noticed several clear limitations of these tools, as follows:

Global metrics: Comparing accuracies across classifiers is certainly useful in providing a simple ranking between them (Tab. 2). However, as already indicated there, such metrics are too aggregated for allowing more fine-grained interpretations. For instance, Tab. 2 shows us that DeepSea (ISIIS), AlexNet (NSDB), and AlexNet (ImageNet) are very close to each other, differing only by a few percentage points of accuracy. This does not tell us how these classifiers actually differ from each other, leaving the possibly wrong impression that they behave similarly. We argue that these limitations are inherent for all other global metrics besides accuracy.

Confusion matrices: This instrument refines the insights provided by global metrics by providing information at class level (Fig. 3.6). However, following the terminology introduced in Sec. 2.2.4, confusion matrices are class-centric techniques: They show which classes are easy (or hard) to classify, but not which are the kinds of observations (in a class or several classes) that actually create classification problems. Also, they do not show whether certain classes are hard to classify because their instances are, for example, quite similar to each other.

Individual observations: Figure 3.7 takes the opposite extreme from the above two plots. Here, detailed information is presented for individual instances (observations). This lets us see which are the classifiers that succeed, or fail, in handling these specific instances; and also allows us to visually compare the instances themselves to infer possible causes for the confusion. However, this visualization is manually created, by having the user select (by browsing) a set of interesting samples to examine. In other words: Once we know which are interesting samples, we can visualize them in detail this way, but how to find the interesting samples in the first place? Moreover, this visualization is not scalable, being able to accommodate a few tens of images at best. Finally, while users can visually compare images to elicit similarities, the visualization provides no insight in how the *classifiers* actually see the images as being similar or not due to their features.

Class histograms: Figure 3.8 presents a final visualization of our results, this time in terms of per-class accuracy for all classifiers, all classes. At a high level, the insights conveyed by this visualization are quite similar with those from confusion matrices (Fig. 3.6). The barchart design, however, allows for an easier comparison of accuracies both across classes and across classifiers. Still, just as the confusion matrices, this visualization does not convey any observation-centric or feature-centric information.

Summarizing the above, we see that classical visualization techniques, albeit useful in conveying several insights, and answering several questions, related to classifier engineering, also have clear limitations. In particular, (a) such techniques convey little insight on how observations and/or classes resemble each other from the perspective of their features; and (b) they do not convey any information on where, in the feature space, a classifier actually “flips” to change the inferred class. The visualization techniques presented in the following chapters aim to address these limitations.

4

CONSTRUCTING DECISION BOUNDARY MAPS

In Chapter 3, we have presented an end-to-end application of classifier engineering using deep learning. While the presented results show that, given suitable engineering, one can construct a classification model giving good results (in terms of overall and per-class accuracy), the *process* of classifier engineering itself raises several questions. First, the *visualization* techniques used in this process – tables, matrix plots, bar charts, and individual samples – provide only limited, and typically aggregated, insights on the actual behavior of the models at hand. In other words, we see *what* the overall behavior of a classifier may be, but not *why* the classifier behaves that way. Secondly, in cases where the performance of the classifier is deemed insufficient, these tools do not tell us *how* to improve the performance (or help us in the improvement process). We claim that, for both above tasks, different, and more fine-grained visualizations, can help.

Related to our central research question stated in Chapter 1, we argue that visualizing the *decision boundaries* of a classifier (Sec. 1.2) can be one such instrument. More specifically, being able to better understand how decision boundaries emerge in a feature space, and how they are affected by the training process, can (a) convey useful information on the effectiveness of the training process, and then (b) help the engineer in taking decisions that steer the training towards the desired optimal goal. For example, a visualization of such decision boundaries can highlight areas in the feature space where the boundaries are not suitable, *e.g.*, too tortuous, fuzzy, uncertain, too close to certain samples, or cutting wrongly through samples of different classes. Seeing this, the user can act upon the training data and/or classifier hyperparameters, triggering the recomputation of new boundaries. By monitoring this "computational steering" loop, the user can arguably drive the classifier construction to the desired outcome.

To address the above goal, we propose in this chapter a method to actually *construct* visual depictions of decision boundaries and decision zones. For brevity, we refer to both of these next as decision boundary maps. We proceed as follows. Section 4.1 formulates the construction of decision boundary maps in an image-based (dense visualization) setting. Section 4.2 details the actual techniques used to construct these maps. Section 4.3 shows several results of our proposed technique for different

---

This chapter is based on the following publication:
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

datasets and classification models. Section 4.4 discusses our technique. Section 4.5 concludes this chapter.

4.1 DENSE MAPS

Let us revisit the core idea behind the definition and construction of decision boundaries and decision zones for a classifier. As already introduced in Sec. 1.2, decision boundaries and decision zones are, typically, visualized only *implicitly*. As shown in Fig. 1.1, this can be done by showing a scatterplot of a classified dataset (constructed by using dimensionality reduction, see Sec. 2.2.2), color-coded next by the class values inferred by the model. The decision zones are then *implicitly* perceived as areas of densely packed same-color points in the projection. By implication, the decision boundaries are imaginary (that is, not explicitly drawn) curves that separate such decision zones, *i.e.*, pass between points having different colors. This process of imagining the decision zones and their boundaries is clearly error prone, as the user is forced to imagine the actual trajectory of the high-dimensional decision boundaries purely based on the distribution of colors in a 2D scatterplot.

We propose to enhance this perception by drawing the decision boundaries by first coloring each pixel in the 2D projection space according to the output of the classification function for the high-dimensional point that corresponds to the respective pixel. We call the resulting images *decision maps*. The process of constructing such maps will effectively "fill in" the blank areas that exist between projected samples in current scatterplots, thereby making the position of the decision boundaries explicit and pixel-accurate. Additionally, when high-dimensional points having different classes project to the same pixel, we will indicate this by suitably mixing the class colors of the respective points, and thereby indicate a fuzzy region of the classifier – that is, a high-dimensional data region in which the classifier assigns different labels. Besides making the decision boundaries explicit, the dense map produced by the above method will give a hint to the user on what one should expect when the classifier is given samples *outside* of the training dataset. Those hints could in turn give the user knowledge about where his/her classifier is uncertain; the user could react by generating more samples, targeted on these uncertain regions, to overcome this deficiency. In turn, this will have the effect of shifting the decision boundaries as more training samples are added. The process loops in a visual analytics fashion until the user is satisfied with the achieved decision boundaries and/or the corresponding classifier accuracy.

Let us now formalize the process of constructing decision maps. Let $D \subset \mathbb{R}^n$ be the data space input by a classifier, *e.g.*, the space of all handwritten digit images for a MNIST-like problem. Let $f : D \rightarrow C$ be a classifier function, where C is a categorical domain containing the various labels assigned by f. For the MNIST example, C is the set of

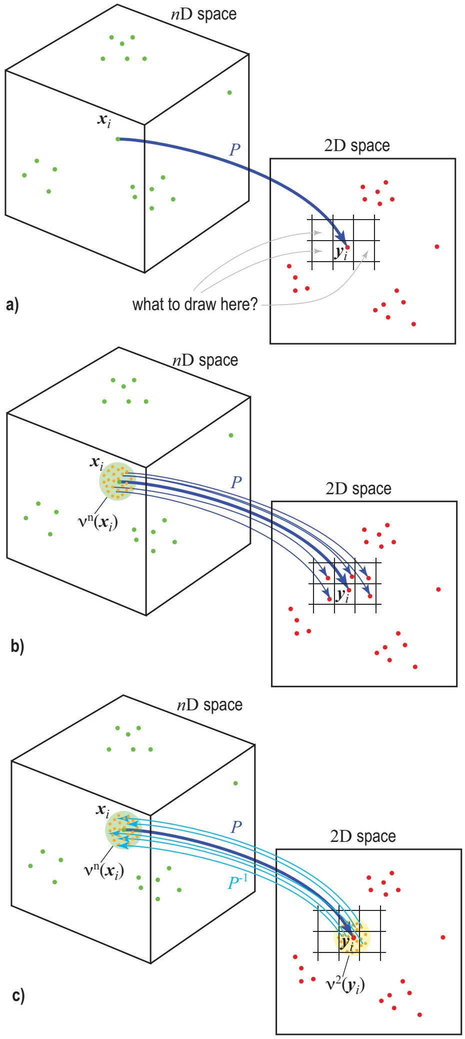


Figure 4.1: Challenge of visualizing decision zones and boundaries represented implicitly in a scatterplot (a). Dense map construction by scattering (b) and gathering (c) approaches.

digits from 0 to 9. The function f is constructed via a so-called training-set $S_t \subset (D \times C)$ and next extrapolated to the entire data space D. In this setting, f can be seen as partitioning D into $|C|$ decision zones D_i, so that $\bigcup_i D_i = D$; $D_i \cap D_j = \emptyset, \forall i \neq j$; and all points in a given zone D_i have the same label in C. Consequently, decision boundaries are precisely the boundaries of the compact zones D_i.

Typically, decision zones and their boundaries are not directly computed nor visualized in the above setting. Rather, one selects a finite point-set $S_T \in D$, *e.g.*, a test set for the classifier. Then, one projects this point set to 2D by using any suitable projection method, and next color-codes the resulting scatterplot by the label values $f(\mathbf{x}|\mathbf{x} \in S_T)$ (Fig. 4.1a). As already explained, this is only a sparse sampling of the actual classifier behavior, and the user has to mentally reconstruct (or guess) how the decision zones and their boundaries look like from the scatterplot $P(S_T)$.

Ideally, a (two-dimensional) dense map that depicts the behavior of the classifier f would (a) map every point $\mathbf{x} \in D$ in the classifier's data space to some point $\mathbf{y}$ in 2D, so that the entire space D is shown; and (b) depict the value $f(\mathbf{x})$ at the point $\mathbf{y}$. However, achieving this is practically impossible in most cases for two reasons: Desiderate (a) above would imply that we are able to map the entire nD *space* in D to 2D *space* in some meaningful way[1]. Desiderate (b) above could imply that we need to depict different labels $c_1 \in C, c_2 \in C, c_1 \neq c_2$ at two 2D points $\mathbf{y}_1 \in \mathbb{R}^2, \mathbf{y}_2 \in \mathbb{R}^2$ which are closer to each other than the available screen resolution.

Since creating a dense map that satisfies both conditions (a) and (b) above is not possible in general, we resort to a sampling-based approximation. Let $I \subset \mathbb{R}^2$ be the (compact) domain represented by the screen space in which we visualize the scatterplot – this corresponds to the interior of the black rectangle in Fig. 4.1a (right). Ideally, the image of D through P, denoted $P(D)$, would cover the entire domain I, thereby delivering (at least) one label $f(\mathbf{x})|\mathbf{x} \in D$ for any pixel $\mathbf{y} \in I$.

To construct a dense map that covers all pixels of I, two approaches can be taken. In the following, let T be and finite set of sample values from D. We call our two approaches gathering, respectively scattering, following the terminology used in image processing for computing convolutions of images (see *e.g.* [172]). These are as follows.

Scattering approach: In this approach, we can create new observations $\mathbf{x}_j \in D$ by densely sampling the neighborhoods $\nu^n(\mathbf{x}_i)$ of all samples in T, computing their labels $f(\mathbf{x}_j)$, projecting $\mathbf{x}_j$ to 2D, and color-coding the pixels $\mathbf{y}_j = P(\mathbf{x}_j)$ by the labels $f(\mathbf{x}_j)$ (Fig. 4.1b). This method of *scattering* data from $\mathbb{R}^n$ to $\mathbb{R}^2$ does not guarantee that all pixels of I get covered, unless potentially very large neighborhoods ν^n are used, which is computationally expensive. More specifically, a scattering strategy is impractical as the number of samples needed to cover the data space D grows exponentially with the number of dimensions n. As it is usual for ML datasets to contain hundreds or even thousands of dimensions, it is infeasible to generate such amount of data by brute force.

---

[1] Formally speaking, we can construct many continuous mappings that achieve the above. However, in general, these would not allow one to decode properties of nD points, such as similarities, from the 2D representation, thus would be not useful in practice.

Gathering approach: An alternative to scattering is to supersample I with $N \geq 1$ samples per pixel $\mathbf{y}$; find the point $\mathbf{x} \in D$ that projects to $\mathbf{y}$, by using an (approximate) inverse projection function P^{-1}; and color $\mathbf{y}$ to summarize all labels of points that project onto that pixel (Fig. 4.1c). This *gathering* method guarantees that every pixel depicts information from at least N high-dimensional samples, with a limited computational cost. Note that gathering strategies are also preferred to scattering ones in image processing, as exemplified by [172].

4.2 DECISION BOUNDARY MAP CONSTRUCTION

Given its computational advantage we choose for our goal the gathering strategy, and implement it as follows. Let N be the user-specified minimal number of samples per pixel desired. Larger N values increases the confidence of our visualization, as we have more information to decide the value of each pixel. Given a sparse labeled set of samples $\mathcal{X}$, we first compute its scatterplot $P(\mathcal{X})$. This delivers $n(\mathbf{y})$ labels per pixel $\mathbf{y}$, where $n(\mathbf{y}) = 0$ for most pixels, given the above-mentioned sparsity. To ensure our target of N samples per pixel, we synthesize $\max(N - n(\mathbf{y}), 0)$ 2D points randomly spread over each pixel $\mathbf{y}$, and compute their $\mathbb{R}^n$ counterparts using P^{-1}. Pixels which are already densely covered by points in $P(\mathcal{X})$ need fewer additional samples, whereas pixels not at all covered by $P(\mathcal{X})$ receive N additional samples each.

At this point, every pixel is covered by at least N labels. We next encode the sample density, classifier confusion, and classifier decision at each pixel, as follows.

Decision: We define the decision d for a pixel $\mathbf{y}$ as the majority class label for all samples $\mathbf{y}_i$ in $\mathbf{y}$, *i.e.*

$$d(\mathbf{y}) = \text{argmax}_{k \in C} \sum_{\mathbf{y}_i \in \mathbf{y}} [\![f(P^{-1}(\mathbf{y}_i)) = k]\!], \tag{4.1}$$

where $[\![\cdot]\!]$ denotes Iverson's bracket. We encode d in the hue $H(\mathbf{y})$ via a categorical color map, as follows. For each class label $k \in C$, we define a basic hue $H_T(k)$ and a slightly lighter version thereof $H_{synth}(k)$. If a pixel $\mathbf{y}$ has the majority label k, and there are points in the original input dataset T that project over $\mathbf{y}$, we set $H(\mathbf{y}) = H_T(k)$, otherwise, we set $H(\mathbf{y}) = H_{synth}(k)$. This way, we can distinguish between pixels covered by the scatterplot $P(T)$, which use $H_T(k)$, and pixels for which we needed to synthesize additional samples, which will use $H_{synth}(k)$.

Confusion: We define the confusion $c(\mathbf{y})$ for all samples of a pixel $\mathbf{y}$ as the ratio between the number of samples of the class label having most instances over $\mathbf{y}$ and the total sample count for that pixel, *i.e.*,

$$c(\mathbf{y}) = \frac{\max_{k \in C} \sum_{\mathbf{y}_i \in \mathbf{y}} [f(P^{-1}(\mathbf{y}_i)) = k]}{n(\mathbf{y})}. \tag{4.2}$$

Higher c values indicate more consensus for the samples over a pixel, whereas lower values indicate that the respective pixel is close to, or on, a decision boundary. We encode confusion into the saturation $S(\mathbf{y})$: Colorful pixels indicate areas where f chooses consistently a single label (depicted by hue, see above), whereas gray pixels indicate areas close to decision boundaries.

Density: We define the density of samples, ρ, for a pixel, $\mathbf{y}$, as the total number of samples covering this pixel's extent. These can be either samples in T or additionally generated samples created as described above. Visualizing ρ is useful as it tells us which dense-map areas have more information (samples), thus, we can be more confident about. Let ρ_{max} be the highest sample density over all pixels of I. We could directly encode ρ into the value (brightness) $V(\mathbf{y}) = \rho/\rho_{max}$. However, a problem of this design is that inherently darker hues, *e.g.* blue, will exhibit less brightness variation than brighter hues, *e.g.*, yellow, so density variations for the dark-hue labels will be hard to see. Hence, we choose to encode ρ in both brightness and saturation, as follows. First, we normalize ρ to $[0, 1]$ by computing

$$\overline{\rho} = \max\left(\frac{1}{20}\frac{\rho}{\rho_{avg}}, 1\right),$$

where ρ_{avg} is the average sample density over all pixels in I. Next, if $\overline{\rho} \in [0, 0.5]$, we compute V by linearly interpolating between a low brightness value $V_{min} = 0.1$ and the maximal $V = 1$. If $\overline{\rho} \in [0.5, 1]$, we set $V = 1$ and compute the saturation S by linearly interpolating between full saturation $S = 1$ and a low saturation $S_{min} = 0.2$. The net effect is that low densities will appear as darker hues; average densities will show the full brightness of the corresponding hue; and high densities will increasingly brighten the respective hue towards white. The values V_{min} and S_{min} are chosen so that we do not reach pure black or pure white, so the user does not confuse the emerging colors with those corresponding to maximal confusion values (grays). The decision zones of f will appear as "shaded cushions" whose domes indicate high density areas, akin to the results shown (by a different implementation and for a different goal) in [131].

Figure 4.2 summarizes the HSV color synthesis proposed above to encode decision, confusion, and density for a class label mapped to the hue red (for illustration purposes). Along the y axis, we see how brightness increases to map higher density values. When the normalized density $\overline{\rho}$ is equal 0.5 and confusion is zero, we get a pure fully saturated red color. Lower density values map to darker reds, while higher densities map to increasingly whitish reds. Along the x axis, we see how saturation decreases to map increasing confusion values. When confusion is maximal, we only see gray tints.

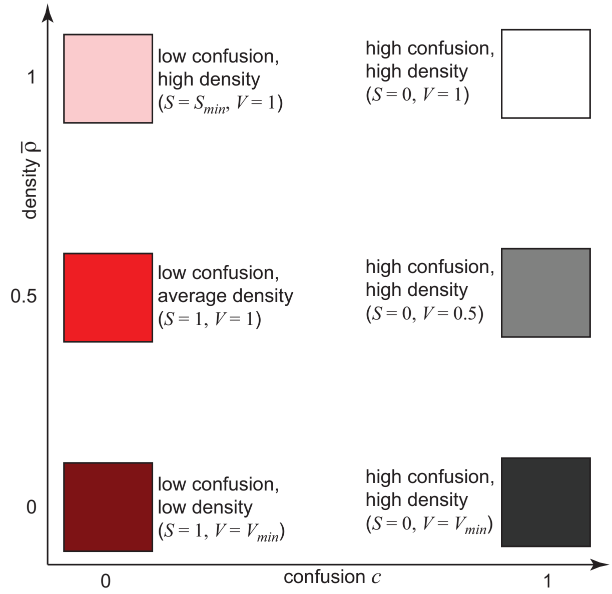


Figure 4.2: Color scheme encoding decision, confusion and density values.

4.2.1 *Parameter setting*

Our proposed dense map depends on two free user parameters: the resolution R of the target image I and the minimal desired number of samples per pixel N. We next explore the insights delivered by varying these parameters on a simple two-class dataset. This dataset is a subset of the well-known MNIST benchmark [75], created by keeping only the images of the digits 0 and 1 (for further details on MNIST, see Sec. 4.3.2). For illustration purposes, we trained a Logistic Regression classifier (f) on this dataset, achieving a 99.8% accuracy. Any other classifier could be used – leading, of course, to different dense maps showing the behavior of that classifier. For projection, we used LAMP [66].

Figure 4.3 shows the impact of varying R and N on the dense map. Several observations follow, first and foremost, we see that the differences between dense maps for different parameter values are small and, more importantly, vary continuously with the parameters. This tells that our dense map construction is stable with respect to parameter choice, which is very important for its practical usability. Secondly, we see how the brightness bump, visible on the top-left image ($N = 1$, $R = 50 \times 50$ decreases as either R or N increase. This is expected, and interpreted as follows: for low N and R values, density variations of the raw *input* dataset D are visible, since there are no additional samples needed to construct the dense map. For our example, the red brightness bump tells that the “red” class samples are overall much denser than the “blue” class samples. Such images can be seen as a direct, depiction of D.

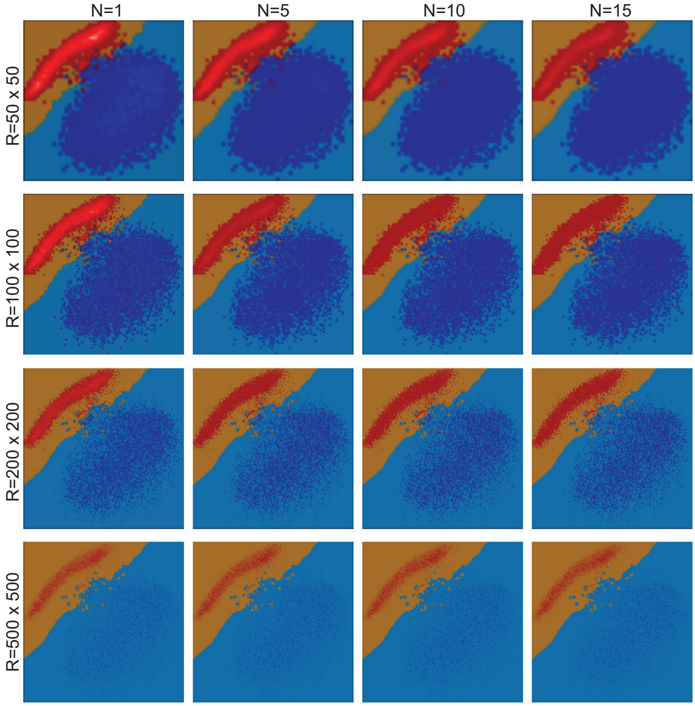


Figure 4.3: The effect of varying both resolution R (rows) and number of minimum number of samples per pixel N (columns).

As either N or R increases, the number $n(\mathbf{y})$ of samples in D per pixel $\mathbf{y}$ will decrease becoming eventually lower than N (the minimal number of desired samples per pixel), so we need to synthesize additional samples. When adding these extra samples, the overall spatial density of samples over the image becomes relatively constant, converging to N in the limit, so we see less brightness variation. We can interpret such high-N, high-R images as converging to the actual continuous decision boundaries in the limit. As R increases, we also see how the decision boundaries become more refined, showing more fine-scale details. Separately, the fact that we see few desaturated (gray) colors in the images tells us that the depicted classifier is quite consistent – that is, it assigns the same class to close samples.

To better understand confusion zones, Fig. 4.4 shows a zoomed-in view on the same dataset, but uses a simpler color coding than Fig. 4.3 – hue encodes the majority class label per pixel $d(\mathbf{y})$ (Eq. 4.1) and saturation encodes confusion $c(\mathbf{y})$ (Eq. 4.2). Hence, whitish pixels indicate zones where the classifier has a high confusion. As we increase the sampling density N, confusion bands appear more pronouncedly along the red-blue decision boundary, which is expected, since close to this boundary the classifier needs to change decisions. We also see small confusion areas *within* the compact blue zone, which indicate that this classifier has likely "drawn" the decision boundary in a too simple and inaccurate way.

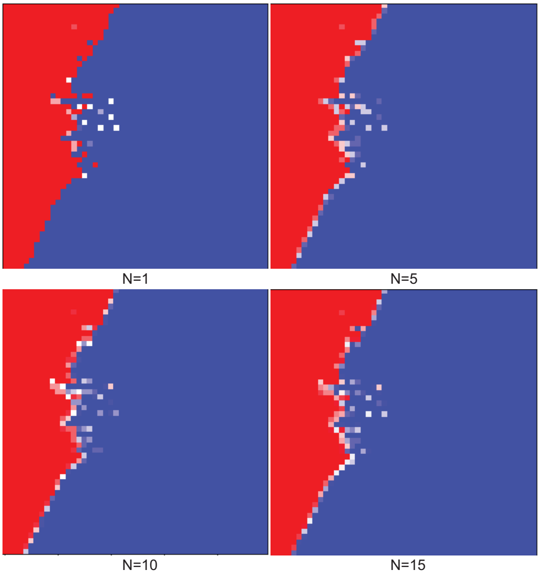


Figure 4.4: Confusion zones (bright pixels) along the decision boundaries as function of minimum samples per pixel N.

4.2.2 *Implementation details*

We implemented our dense maps in Python using t-SNE [82] and LAMP [66] for the direct projection P and iLAMP [124] for the inverse projection P^{-1}. LAMP and iLAMP are simple to implement, and, as shown in several works, achieve higher accuracies in preserving distances than similar-type projection methods [66, 85]. In contrast, t-SNE has a better ability to separate high-dimensional clusters [82] than LAMP. LAMP and t-SNE are further compared in Sec. 4.3.2.

4.3 EXPERIMENTAL RESULTS

We illustrate our technique by applying it to two high-dimensional datasets, four classifiers, and two dimensionality reduction techniques, as follows.

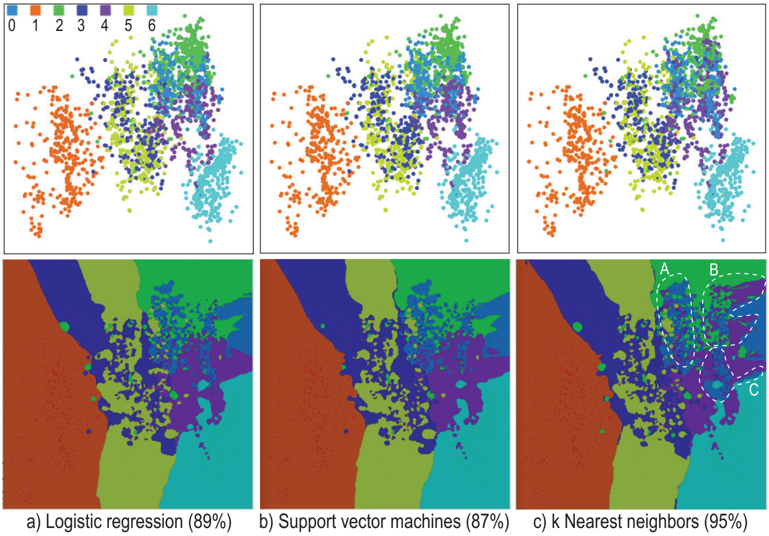
4.3.1 *Segmentation dataset*

The Image Segmentation Dataset [32] contains 2310 image instances with 19 features each, divided into 7 classes. Features measure image attributes, *e.g.*, color intensity mean, contrast, hue, and saturation. Classes relate to types of outdoor images, *i.e.*, brickface, sky, foliage, cement, window, path and grass. We trained three different classifiers, *i.e.* Logistic Regression (LR), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN), all implemented in *scikit-learn*[104], on this dataset, with the aim of comparing their decision boundaries. As parameters, we used the default ones in *scikit-learn*, except radial basis functions (SVM), and $k = 5$ nearest neighbors (KNN). Data was split into 70% training samples and 30% test samples. The obtained accuracies were 89% (LR), 87% (SVM), and 95% (KNN).

Figure 4.5 (top row) shows three LAMP projections for the three classifiers, each categorically colored by the training and test-set labels. This is a typical way that ML practitioners use to assess how the classifiers “divide” the data space into different zones for different classes. From these images, we only see small-scale differences between the three classifiers. Moreover, as already explained, such images are subject to occlusion and overplotting. Also, from these images, it is not clear what a classifier would decide for a sample which is relatively far away from existing ones.

The dense maps, generated at a resolution $R = 500^2$ pixels, and with a minimum number of $N = 5$ samples per pixel, attempt to overcome these issues (Fig. 4.5, bottom row). They show us several insights. First, there is no overplotting in these images, so we are sure that each pixel carries the exact information pertaining to the samples that fall over it. Secondly, the differences between the decision zones corresponding to the seven classes are now much easier to see. For instance, for all classifiers, class 1 (orange) has a quite smooth and clear decision boundary touching mainly classes 3 (dark blue) and 5 (yellow-green). However, subtle differences between the classifiers also show up – for instance, the class-1 decision zone of LR contains a few isolated islands for class 2 (green) and class 3 (dark blue). These islands are different for SVM and KNN. Separately, we see that the decision boundaries for the other classes, most notably class 0 (blue) and class 5 (yellow-green) are much more jagged, for all classifiers. Verifying the actual classification results shows, indeed, that instances in these classes are harder to classify than in *e.g.* class 1 or class 6.

Finally, we see some interesting differences between the dense map of KNN and the other two (Fig. 4.5, white stippled lines): the decision boundary of class 4 (purple) is visibly stretched upwards in zone A for KNN, whereas for LR and SVM, the purple zone is much smaller and does not protrude upwards through the class-0 (blue) area. Similarly, the decision boundary of the same class 4 protrudes significantly upwards in the green area in zone B for KNN, but not for the other two classifiers. Finally, the decision boundary for class-0 (blue) protrudes significantly downwards in the purple area for KNN, but not for the other two classifiers. Note that these differences cannot be explained by the projection, since we use the same projected points for all three classifiers. Overall, the decision boundaries of KNN show larger, and more mixed, per-class areas, except for class 1. This explains both the increase in accuracy of KNN *vs* the other two classifiers, but also for which regions (types of images) of the data space these differences occur. Note, again, that spotting such differences using only standard color-coded projection scatterplots is very hard, or even impossible.



a) Logistic regression (89%) b) Support vector machines (87%) c) k Nearest neighbors (95%)

Figure 4.5: Projections (top) and dense maps (bottom) for *segmentation* dataset, three classifiers.

4.3.2 *MNIST dataset*

Our second dataset, MNIST, is a standard dataset in ML consisting of 70K handwritten digit images, commonly employed to evaluate the performance of machine learning image classifiers, split into 60K training and 10K testing images [75]. Each 28 × 28 pixels grayscale image can be interpreted as a point in a 784-dimensional space. We use this dataset to explore two other questions, as follows.

First, we show that dense maps can also be used for deep learning classifiers, apart from the more classical ones such as LR, SVM, or KNN. For this, we built a simple Convolutional Neural Network (CNN) composed of two convolutional layers, one max-pooling layer and two densely connected layers. This CNN was implemented and trained using Keras [23], and obtained an accuracy of 99.2% on the test data after 14 training epochs. We next computed a 2D projection from a subset of 2000 samples of the training dataset using t-SNE (Figure 4.6). We did

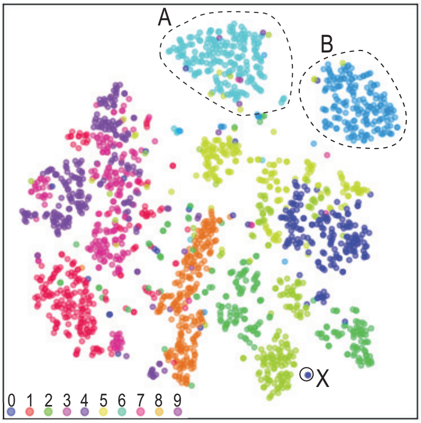


Figure 4.6: MNIST projected using t-SNE, color coded by class labels.

not use the full training dataset as t-SNE is quite slow (quadratic in the number of data points). If we had only this projection to analyze the decision boundaries assigned by the classifier, what would then be the conclusions to be drawn?

For instance, consider the top-right class-2 (green) and class-0 (blue) groups (marked A and B in Fig. 4.6). These appear equally well separated from the rest of the projection, equally compact, and have a similar (low) number of other-class points embedded in them. As such, with only the sparse projection information, we would likely conclude that the decision areas and boundaries for these two classes are quite similar. Likely, the user seeing this projection would draw the decision zones corresponding to A and B much like the dashed lines shown in Fig. 4.6. Let us look at the dense map for this dataset. Figure 4.7 shows it, at a resolution $R = 300^2$ pixels, computed for four different values of the per-pixel sample density N. We see now that the decision zones and boundaries of class-2 and class-0 are, in fact, much more complex than we could infer from the scatterplot. In particular, we see a non-negligible number of small "islands" corresponding to other labels than 0 and 2 embedded in the zones for these two labels. Also, we see that the class-0 zone is much more compact than class-2 – it contains a single small island for class 5 (Fig. 4.7, marker C).

Separately, let us consider the question of what happens close to outlier training samples. Take, for instance, the point marked X in the scatterplot (Fig. 4.6). What label would be assigned to a digit image that projects there? We see that we have an isolated class-3 outlier and the closest samples are the relatively large class-8 group (orange). So, based on the scatterplot, one would reckon that some class-3 decision boundary surrounding the outlier point will be created. However, what is the exact shape and size of this decision boundary? We cannot answer this question using only the scatterplot. The dense map gives us precisely this answer: the point X falls within an "island" decision zone that corresponds to class 3 (dark blue). This island is quite large, so it tells us that the impact of a *single* outlier in the training set is important in such sparsely-sampled areas. Note that this is not an approximate result: our method indeed synthesized a group of samples that project around (and on) the point X, ran it through the classifier, and obtained class 3 as a result. Moreover, we see that the dense maps are practically identical for different per-pixel sample densities, which increases the confidence that the class-3 island we see is indeed there. We could not have obtained this insight using the scatterplot only.

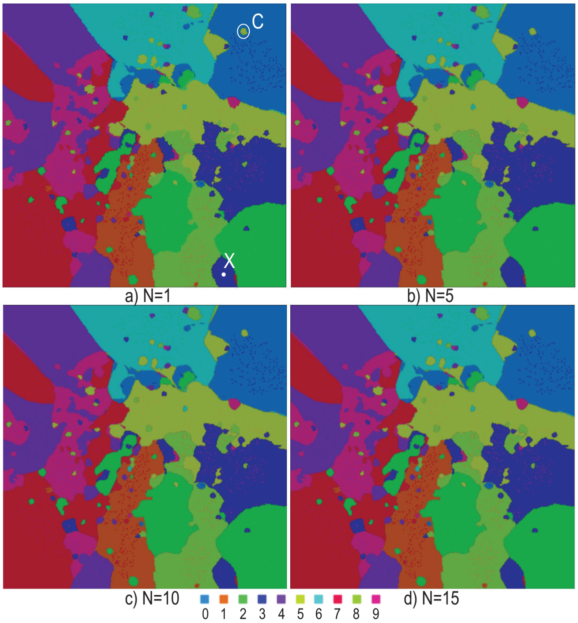


Figure 4.7: Dense maps for MNIST dataset classified by CNN, different sample density values N.

Another use of our dense maps is in showing how the decision boundaries *change* during training of a classifier. Figure 4.8 shows four such dense maps, for four different epochs (E) of training the CNN for the MNIST dataset, using stochastic gradient descent, with a learning rate of 0.001. For the first epoch (Fig. 4.8a), we see how the decision boundaries are highly jagged, while clear decision zones are mainly visible for the outlier samples. This confirms the insight that during a deep neural network training, outliers are handled the easiest, as surrounding them by decision boundaries is far easier than "drawing" such boundaries through a compact area of very similar samples [116]. From epoch 5, decision boundaries get significantly more refined in the central dense-sample region. The dense maps for epochs 10 and 15 clearly show how the training converges, as the decision boundaries basically stabilize.

The dense maps correlate very well with the testing accuracies reported in Fig. 4.8. Such images generalize the simple 2D animations of 2D dataset classifications provided by TensorFlow [1] to nD datasets and arbitrarily complex networks. They help directly seeing when further training does not bring added value (in our case, from $E = 10$ onwards).

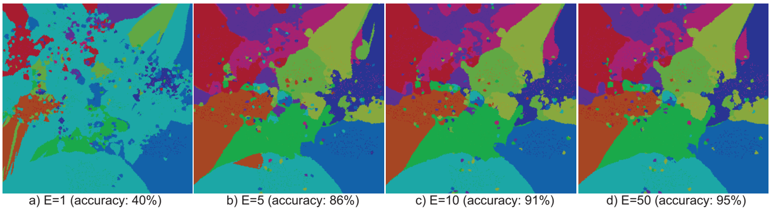

a) E=1 (accuracy: 40%) b) E=5 (accuracy: 86%) c) E=10 (accuracy: 91%) d) E=50 (accuracy: 95%)

Figure 4.8: Dense maps for four training epochs E, CNN classifier, MNIST dataset.

Finally, let us consider the projection algorithm choice. In Sec. 4.2.1, we have shown that LAMP is a good choice for a simple two-class dataset. Above, we have shown that t-SNE works well for the high-dimensional MNIST dataset. We consider the same MNIST dataset (and classifier), but use LAMP for the 2D projection instead of t-SNE. The resulting projection (Fig. 4.9a) shows clearly more class mixing than the t-SNE projection (Fig. 4.6). The explanation follows: t-SNE aims to preserve the high-dimensional nearest *neighbors* in the projection. Also, t-SNE pre-processes the data by PCA dimensionality reduction prior to projection, to make the 2D embedding task easier [82]. In contrast, LAMP aims to preserve high-dimensional Euclidean *distances* between points. So, for hundreds of dimensions (like the 784 ones in MNIST), LAMP yields far less cluster separation in the projection, even if the high-dimensional data is well separated. A poor-separation projection leads, next, to a dense map showing fragmented decision zones with complex borders (see Fig. 4.9b, which uses the same N and R values as Fig. 4.7d). So, we conclude that for low-dimensional datasets, LAMP and t-SNE are comparably good (with LAMP being significantly faster); for high-dimensional datasets, t-SNE should be definitely used instead of LAMP.

4.4 DISCUSSION

We discuss next several important aspects of our image-based visualization of classifier decision boundaries.

Genericity: Our dense maps work for any classifier and dataset, as long as the data can be represented by a feature vector in $\mathbb{R}^n$, so that we can project such features via generic methods such as LAMP or t-SNE, respectively invert the projection via iLAMP. No specific

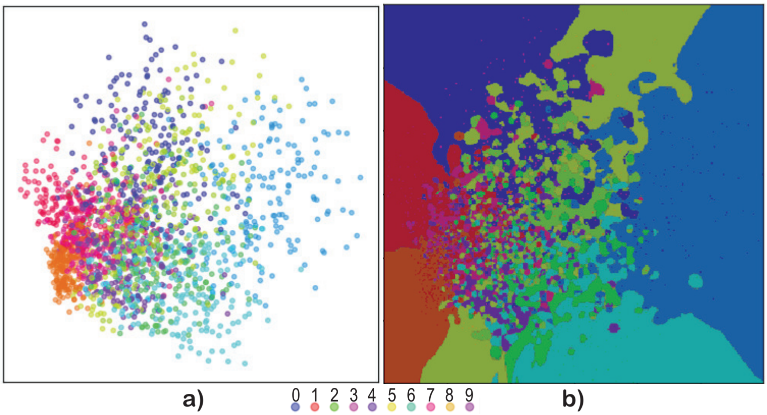


Figure 4.9: LAMP projection and dense map for MNIST dataset, CNN classifier.

constraints on data dimensionality n, data type, classifier internals, or classifier training process, exist. For instance, we can visualize the decision boundaries of an insufficiently trained classifier and compare them with those of a better trained one, to tell us how training shifts these boundaries (see the example in Sec. 4.3.2).

Robustness: We have shown that our method is robust even for sparsely-sampled data spaces (limited number of training samples). Our dense map construction guarantees a user-specified number of labeled samples N per pixel, at a user-given resolution R. As N increases, every point of the 2D image becomes equally densely sampled, meaning that we also have the same confidence everywhere on our dense maps.

Projection: Our dense maps obviously depend on the quality, of the projection being used. A projection that, for instance, does not respect well high-dimensional distances or neighborhoods will also yield an artificially confused dense map. Yet, two key observations must be made. First, this (well known) limitation of projections applies equally to using scatterplots when visualizing a classifier's results, so our dense maps do not add any extra problem here. Secondly, the projection space should be seen as an *abstract*, and not Euclidean, space, in which decision zones are depicted. That is, *topological* tasks like finding the neighbors of a decision zone along its boundary, finding islands, and finding confusion zones, can be completed well even if a projection doesn't perfectly preserve point neighbors and/or inter-point distances. All our experiments showed that t-SNE is a good projection in this respect, in line with earlier findings in the same direction [82, 83, 116].

Limitations: Our current implementation cannot handle tens of thousands of points at interactive rates. The reason hereof is the already-mentioned high computational complexity of t-SNE. Yet, recent t-SNE accelerations could be used [106, 107], when such techniques become publicly available. Also, we need to compute the projection only *once* for a given dataset. A separate limitations regards the methods used to construct dense maps. So far, we only tested two projection methods (t-SNE and iLAMP) and one inverse projection method (iLAMP). How dense maps would differ when computed using other method combinations is a topic to be studied separately in Chapter 5.

4.5 CONCLUSION

We have presented a technique to visualize decision boundaries of arbitrary machine learning classifiers. For this, we propose an image-based approach where every pixel of the 2D output space is attributed a color to show the exact behavior of the classifier in the corresponding region of the high-dimensional space. For this, we use a combination of direct and inverse dimensionality-reduction methods, and we also propose several visual encodings of the classification result, confusion, and sample density. Our method is simple to implement and can handle any classifier and feature-based dataset with no changes. We demonstrate our method on several datasets, classifiers, and using two different projection techniques.

Several directions of refining the dense maps are possible. As already mentioned, decision maps can be constructed using different combinations of direct and inverse projections. We explore these topics in Chapters 5 and 6. The decision boundary maps can be augmented to show more information, such as explicit misclassification regions, and high-dimensional distances or neighborhoods. This would help understanding *why* a classifier constructed its decision boundaries in a certain way and, thus, help in improving them. We will cover this aspect in Chapter 7. Separately, decision boundary maps can be extended in an active learning approach to propose to the user areas where new labels would be needed to *e.g.* reduce confusion or increase classification accuracy. We study this last topic in Chapter 8.

5

EVALUATING DECISION BOUNDARY MAPS

In the previous chapter, we presented a technique to construct and visualize the decision boundaries and decision zones induced by a given classifier. As described there, our technique depends on two main ingredients – the projection technique used to map the high-dimensional data to the 2D visualization space, and the inverse projection technique used to map points in this visualization space to the high-dimensional data space. Since the resulting decision boundary map images will depend strongly on the choice of these techniques, the question is which are *suitable* combinations of direct and inverse projections that lead to decision boundary map images that are easy to interpret and convey a trustworthy impression of the actual decision boundaries that a classifier has? For instance, a projection technique that preserves well neighborhoods in the data space is, arguably, better for constructing decision boundary maps that one that spreads such neighborhoods all over the projection plane. However, the definition of a "best" projection for our task may depend also on how the projection interacts with the actual classifier whose decisions we want to visualize.

The "design space" of all possible combinations for constructing decision boundary maps is very large, as it involves direct projections, inverse projections, classifier techniques, actual datasets, and hyperparameter values. In related work, Espadoto *et al.* [37] have attempted to analyze a related, but much lower-dimensional, space consisting of the behavior of projection techniques for different types of datasets and hyperparameter values, and for this end they needed to evaluate thousands of combinations. Given that our design space has more dimensions, we will not attempt to densely sample it for evaluation. Rather, we will focus, in this chapter, on gauging the effect of a single dimension, namely the type of direct projection technique used to construct decision boundary maps. As such, we aim next to answer two questions:

1. How do the depicted decision boundaries differ as a function of the chosen projection technique?
2. Which projection techniques are best for a trustworthy depiction of decision boundaries?

This chapter is based on the following publication:
M. Espadoto, F. C. M. Rodrigues, and A. Telea. Visual analytics of multidimensional projections for constructing classifier decision boundary maps. In *Proc. IVAPP*. SciTePress, 2019

5.1 PRELIMINARIES

For easing the reader's burden, we briefly repeat here the construction of decision boundary maps, and associated notations, introduced in Chapter 4. Let $D \subset \mathbb{R}^n$ be a data space of interest in a classification problem. Let $f : D \rightarrow C$ be a classification model that maps from data points in D to some label set C. The model f is constructed using a training set $S_t \subset D$ and tested using a test set $S_T \subset D$, $S_T \cap S_t = \emptyset$. Let P be a projection technique from D to $\mathbb{R}^2$, and let P^{-1} be an inverse projection technique from $\mathbb{R}^2$ to D. A decision boundary map for f is an image I constructed as follows (see also Fig. 5.1a):

For every pixel $\mathbf{y} \in I$, we gather all data samples $\mathbf{x} \in D$ that project into $\mathbf{y}$, and, if their count Y is below a user-prescribed value U, we add to them $U - Y$ synthetically created points $P^{-1}(\mathbf{y}')$, where $\mathbf{y}'$ are random points inside pixel $\mathbf{y}$. Having now $R = \max(U, Y)$ data samples $\mathbf{x}_1, \ldots, \mathbf{x}_R$ for each image pixel $\mathbf{y}$, we color $\mathbf{y}$ the labels $L = \{f(\mathbf{x}_1), \ldots, f(\mathbf{x}_R)\}$ assigned by the model f. Compact same-color areas in I indicate decision zones where the classifier f infers the same label, *i.e.*, reflect the underlying so-called contiguity hypothesis typical in many ML contexts [84]; frontiers separating different colors in I indicate decision boundaries. Few compact zones with simple (smooth) boundaries indicate that the classifier has little difficulty in taking decisions over D. Multiple disjoint same-color zones and/or zones with tortuous boundaries indicate the opposite. Small-size "islands" of one color embedded in large zones of different colors suggest misclassifications and/or training problems.

However, the trustworthiness of our dense map technique heavily depends on the direct (P) and inverse (P^{-1}) projection techniques it uses. Consider, for example, a toy two-class kNN 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 lead to a decision boundary map image that shows two compact decision zones separated by a straight line. 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 simple classifier.

As discussed in Chapter 2, tens of different projection techniques P exist. Which ones are best for constructing decision boundary maps is, however, not evident. To find these, we can use, up to some extent, the analysis of Espadoto *et al.* [37] that quantitatively compared over 40 such techniques against six different quality metrics. However, the respective quality metrics are chiefly aimed at gauging how well a projection succeeds in creating a *scatterplot* that faithfully conveys the high-dimensional data structure. In our case, there is, formally speaking, no scatterplot whose quality we want to maximize, but a dense map image.

Hence, it is not evident that projections deemed best by the survey of Espadoto *et al.* are automatically best for our task.

A second evaluation problem exist in our context: How to gauge the quality of a decision boundary map produced by a given projection technique P? For this, we need some ground truth to compare the map with. In the survey of Espadoto *et al.*, this was (relatively) easy to obtain, since projections were compared with ground truth inferred from the high-dimensional data *points* they represent. In our case, this is not possible, since, except trivial cases, we do not know the shape and position of decision boundaries of classifiers in high-dimensional space.

Given the above, we approach the problem of determining the suitability of projections for decision boundary map construction as a two-phase process, as follows:

1. We construct maps (using all projections) for a simple dataset and two-class classification problem, for which we know how the decision boundaries look like. We next rank the tested projections in terms of how well they generate maps that correspond to our prior knowledge on the decision boundaries for this simple problem;

2. We select a small subset of projections that perform best on the experiment in the first phase, and assess how they behave on more complex classification problems and datasets. Since we do not have ground truth here, we only assess the results qualitatively, in terms of noisiness and fragmentation of the resulting decision zones and boundaries.

5.2 EXPERIMENT SETUP

To answer the two questions stated at the beginning of this chapter, we designed a two-stage experiment to study how dense maps depend on dimensionality reduction (DR) techniques and classifiers, covering a combination of 28 DR techniques and 4 classifiers (Figure 5.1b). The ingredients of this experiment are as follows:

Data: We select two different subsets of the Fashion MNIST [161], a state-of-the-art ML benchmark with clothing and accessory images, which supersedes complexity-wise the traditional MNIST dataset [75]. Both MNIST and Fashion MNIST have 70K grayscale images of 28×28 pixels, split into a training set ($|S_t| = 60$K samples) and a test set ($|S_T| = 10$K samples). The two subsets are as follows:

- S_2: A two-class subset (classes *T-Shirt* and *Ankle Boot*) that we hand-picked to be linearly-separable; the reason for this is that, for such a simple configuration, we know what to expect in the

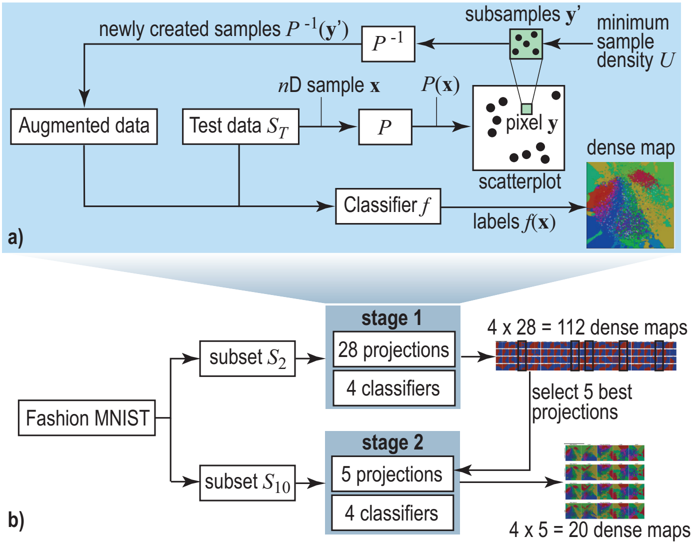


Figure 5.1: Two-phase experiment set-up.

corresponding decision boundary map. Namely, the respective map should (ideally) partition the visual space into two regions separated by a smooth boundary;

- 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 as in [120]: LR, RF, kNN (implemented in *scikit-learn*, using their toolkit's default parameters), and CNN (implemented in *keras*). 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 [70]. These classifiers create very different decision boundaries: At one extreme, LR boundaries are linear (hyperplanes). kNN 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} ($|S_t| = 10800$ samples, $|S_T| = 1200$ samples). We verified that the training yielded good accuracies in all cases (Tab. 3). This is essential to know when we

Table 3: 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
Conv. Neural Network (CNN)	1.0000	0.9080

next gauge the dense maps' ability to capture a classifier behavior (see stage 1 below).

Projections: Table 4 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 inverse projection (P^{-1}), we used iLAMP in all cases, just as in Chapter 4. As selection criteria, we considered well-known projections of high quality (following a recent survey [91][1]), good computational scalability, ease of use (P should come with well-documented parameter presets), and publicly available implementation.

Dense maps: We use a two-stage creation and analysis of dense maps, as follows (Fig. 5.1). 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 (Tab. 3), 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 Tab. 4. 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.

5.3 ANALYSIS OF EVALUATION RESULTS

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

---

[1] The survey of Espadoto *et al.* [37] was not published at the date when we conducted this work. However, upon a detailed comparison, we see that there are no high-quality projections reported in Espadoto *et al.* which our evaluation did not include.

Table 4: Projections tested in phase 1 (Sec. 5.3.2). Projections tested in phase 2 (Sec. 5.3.2) are marked in bold.

Projection	Parameters
Factor Analysis[67]	iter: 1000
Fast Independent Component Analysis (FastICA)[63]	fun: exp, iter: 200
Fastmap[44]	default parameters
IDMAP[89]	default parameters
Isomap[151]	neighbors: 7, iter: 100
Kernel PCA (Linear) [125]	default parameters
Kernel PCA (Polynomial)	degree: 2
Kernel PCA (RBF)	default parameters
Kernel PCA (Sigmoid)	default parameters
Local Affine Multidimensional Projection (LAMP)[66]	iter: 100, delta: 8.0
Landmark Isomap[22]	neighbors: 8
Laplacian Eigenmaps[11]	default parameters
Local Linear Embedding (LLE)[122]	neighbors: 7, iter: 100
LLE (Hessian)[34]	neighbors: 7, iter: 100
LLE (Modified)[167]	neighbors: 7, iter: 100
Local tangent space alignment (LTSA)[168]	neighbors: 7, iter: 100
Multidimensional Scaling (MDS) (Metric)[74]	init: 4, iter: 300
MDS (Non-Metric)	init: 4, iter: 300
Principal Component Analysis (PCA) [67]	default parameters
Part-Linear Multidimensional Projection (PLMP) [101]	default parameters
Piecewise Least-Square Projection (PLSP)[102]	default parameters
Projection By Clustering[100]	default parameters
Random Projection (Gaussian)[30]	default parameters
Random Projection (Sparse)[30]	default parameters
Rapid Sammon[105]	default parameters
Sparse PCA[171]	iter: 1000
t-Stochastic Neighbor Embedding (t-SNE)[82]	perplexity: 20, iter: 3000
Uniform Manifold Approximation (UMAP)[87]	neighbors: 10

5.3.1 *Phase 1: Picking the Best Projections*

As stated earlier, all four tested classifiers yield almost perfect accuracy for the simple 2-class problem S_2 (Tab. 3). Hence, their decision boundaries are “where they should be”, *i.e.*, 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 5.2 and 5.3 show the dense maps for all 28 tested projections *vs* the four tested classifiers, where red and

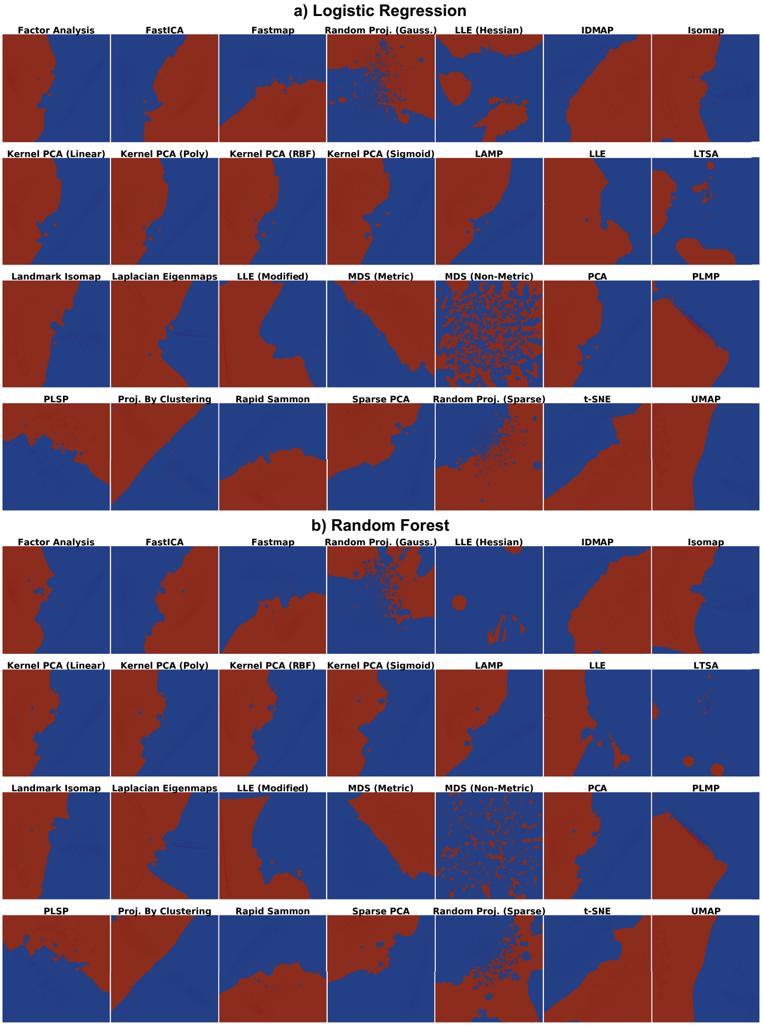
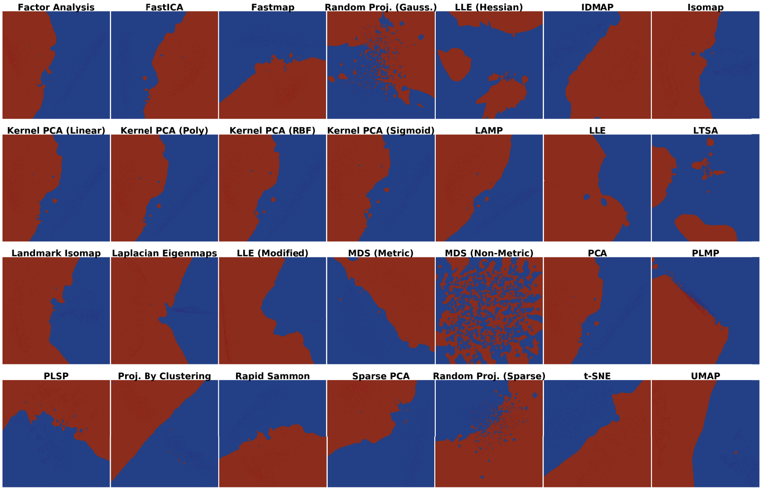


Figure 5.2: Dense maps for Logistic Regression (a) and Random Forest (b) classifiers on the 2-class S_2 dataset, all 28 tested projections.

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.

a) k Nearest Neighbors



b) Convolutional Neural Network

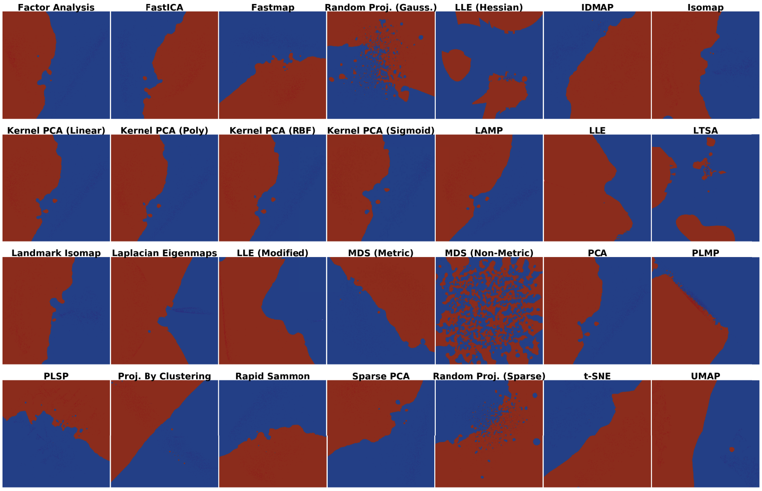


Figure 5.3: Dense maps for k-NN (a) and CNN (b) classifiers on the 2-class S_2 dataset, all 28 tested projections.

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. At this point, it is interesting to remark that, three of the above five methods – Projection by Clustering, UMAP, and t-SNE, were also ranked among the top projections in the independent study of Espadoto *et al.* [37], which used different quality metrics than our work.

5.3.2 *Phase 2: Refined Insights on Complex Data*

We now examine how the five projections selected in phase 1 perform on the 10-class dataset S_{10}, which is a tough classification problem [161]. We already see this in the lower achieved accuracies (Tab. 3). Hence, we expect to have significantly more complex boundaries. Figure 5.4, 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 Fig. 5.4 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 Fig. 5.4 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.

Another salient visual element of the dense maps in Fig. 5.4 is the presence of many small color islands – that is, small-area compact zones of some color (class) surrounded by larger-area zones of another color (class). Let us analyze these in more detail. 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

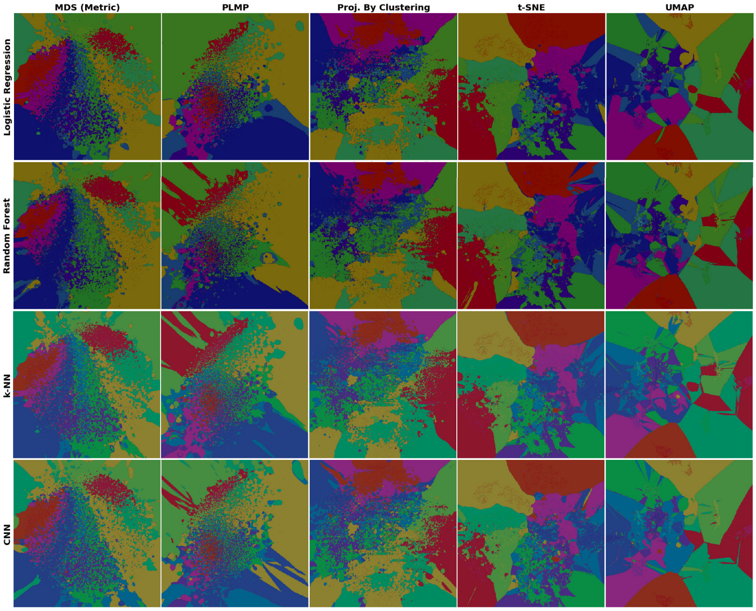


Figure 5.4: Dense maps for all classifiers, 10-class dataset, five best-performing projections.

b) the island *may* exist in D, *i.e.*, there exist very similar samples that get assigned different labels. This case can be further split into
   b1) the island *actually* exists in D, *i.e.* 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, *i.e.*, 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 Fig. 5.4, we plot misclassified points atop the dense map as half-transparent white disks. Figure 5.5 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 Fig. 5.4a, 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 Fig. 5.4b 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. We will present such techniques in Chapter 7.

Separately, we see that, overall, the LR dense maps have more white dots than the CNN ones, which correlates with the lower LR accuracy (Tab. 3). 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 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.

Another finding is that hard samples on the dataset, *i.e.*, the ones located far away from their label group and which appears as "islands" of one color inside another, are easy to spot and if classified correctly, shows that the classifier did a good job on those.

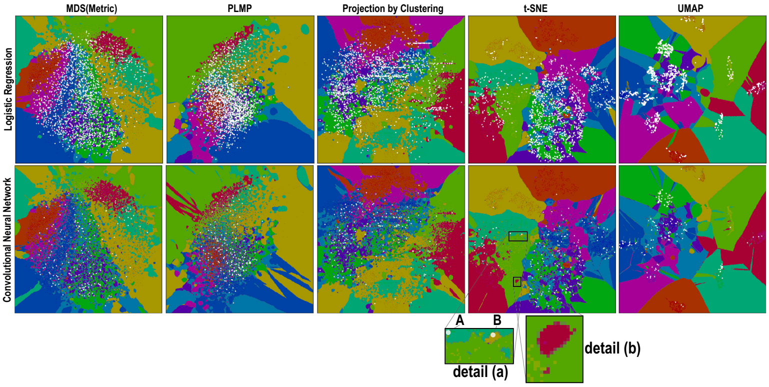


Figure 5.5: Classification errors (white dots) shown atop of the dense maps, LR and CNN classifiers.

5.4 DISCUSSION

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

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. Interestingly, the survey of Espadoto *et al.* [37] on the quality of projection techniques also flags t-SNE and UMAP between the three best techniques, although it gauges a different task and uses different quality metrics. We believe that this is not by chance, but it actually indicates that these two techniques are indeed among the best in existence for a wide variety of tasks.

Influence factors: As mentioned, dense maps depend not only on the direct projection P but also on its inverse P^{-1}. We studied in detail the dependency on P, but only used a single P^{-1} implementation (iLAMP). This is due to the fact that (at the time of doing this work) we were not aware of any other scalable, generic, and publicly-available inverse projection alternative. However, designing such alternatives is an interesting topic. We will cover this point in Chapter 6.

Experiment coverage: Dense maps constructed using projections are a novel technique in high-dimensional visualization. Besides their use discussed here for showing classifier boundaries, they are also used to analyze projection quality [8, 85]. All such maps strongly depend on the projection technique being used. To our knowledge, our current work that evaluates how dense maps depend on the choice of 28 possible projection techniques, is the broadest evaluation of this type in existence. To limit the amount of work required to analyze over hundred 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. This, of course, limits the potentially interesting insights one can find. The same is true for our choice of using a single (though, highly-recognized complex ML benchmark) dataset.

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 [2]. We intend to organically extend this repository with new instances along all above-mentioned dimensions.

In this chapter we have presented a methodology for evaluating the quality of multidimensional projections for the task of constructing 2D dense maps to visualize decision boundaries of ML classifiers. To this end, we have evaluated 28 well-known projections on a two-class, respectively ten-class, subset of a well-known ML benchmark, using four

---

[2] `https://mespadoto.github.io/dbm/`

classifiers often used in practice. Our evaluation shows wide, and to our knowledge, not yet known, differences between the behavior of the studied projections. The closest work to ours that we are aware of is the projection benchmark study of Espadoto *et al.* [37], which evaluates 44 projection techniques from the perspective of quality metrics related to the assessment of *scatterplots* constructed by projecting data. While our task for which we use projections is different (constructing decision boundary maps), and our evaluation criteria are also different (we want decision maps which faithfully reflect prior knowledge on how the decision boundaries look for specific datasets and classifiers), it is interesting to see that both our work and that of Espadoto *et al.* find a common subset of “best” projections, namely UMAP, t-SNE, and Projection by Clustering.

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. On the practical side, our results can be used to drive the selection of suitable projections for other types of dense maps used in high-dimensional visualization. On the methodological side, our workflow can serve as a model for the exploration of a large design space in similar visual analytics contexts.

As already pointed out several times in this chapter, decision boundary maps depend both on the direct and the inverse projection technique used in their construction. In this chapter, we studied the effect of the former. The effect of the latter, including improvements upon existing inverse projection techniques, is the topic of the next chapter.

6

INVERSE PROJECTIONS FOR DECISION BOUNDARY MAPS

The construction of decision boundary maps introduced in Chapter 4 relies on two central techniques – direct projection and inverse projection. In Chapter 5, we studied the effect of the choice of direct projection technique on the resulting maps, and shown that among the studied techniques, t-SNE and UMAP deliver the best results. In this chapter, we focus on the effect of the choice of inverse projection techniques.

The importance of inverse projections to the construction of decision boundary maps has two components, as follows.

Computational effort: Following our usual notation, let D be a finite dataset, *e.g.*, test set, used to assess the working of a classifier f. Let P be the projection technique used to project D to yield a two dimensional scatterplot Y. Let P^{-1} be an inverse projection technique. Finally, let I be an image of N pixels that stores the computed decision boundary map. As explained in Chapter 4, computing this map implies three steps: (a) projecting D to yield the scatterplot $Y = P(D)$; (b) for each pixel $\mathbf{y} \in I$, subsample $\mathbf{y}$ to create R points; (c) for each of these points $\mathbf{y}'$, compute $\mathbf{x} = P^{-1}(\mathbf{y}')$, and next color $\mathbf{y}$ based on the labels $f(\mathbf{x})$. A typical test set D would contain thousands of samples (which need to be projected via P), while a typical decision boundary map would have a resolution N of several hundred thousand pixels. Given that each such pixel is sampled N times, the cost of the entire computation is dominated by the evaluation of f and the computation of P^{-1} at each such pixel sample point. Hence, having an efficient way to evaluate P^{-1} is paramount for using decision boundary maps in a visual analytics interactive setting.

Quality: As discussed in Chapter 5, the quality of decision boundary maps strongly depends on the quality of the underlying projection technique P. Projections which exhibit many false neighbors and/or missing neighbors [85] will "mix up" data coming from unrelated regions in the high-dimensional space when creating the scatterplot, thus lead the undesired effects such as noisy decision boundaries and/or spurious islands in decision zones. By following the same reasoning, it is clear that

---

This chapter is based on the following publication:
M. Espadoto, F. C. M. Rodrigues, N. S. T. Hirata, R. Hirata Jr., and A. C. Telea. Deep Learning Inverse Multidimensional Projections. In *Proc. EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association, 2019

errors in inverse projections P^{-1} would lead to similar problems. Hence, we need high-quality inverse projections for our goal.

In this chapter, we study the effect of inverse projections to the construction of decision boundary maps, with two main contributions. First, we study two existing inverse projection techniques, namely iLAMP[124] (discussed in Sec. 2.2.3.1) and RBF based inverse projection [5] (detailed in 2.2.3.2). Secondly, we propose a new inverse projection method based on deep learning. We compare all three inverse projection methods from the viewpoints of computational speed and quality of resulting decision boundary maps, and show that the neural-network inverse projection – next dubbed NNinv – achieves the best quality and speed from all three studied techniques.

This chapter is structured as follows. First, we detail the construction of our new inverse projection technique NNinv (Sec. 6.1). Having presented this method, we now compare all three inverse projection methods (iLAMP, RBF, and NNinv) using various datasets, classifiers, and direct projection techniques (Sec. 6.2). Section 6.3 concludes this chapter by a discussion of our results.

6.1 INVERSE PROJECTION BY NEURAL NETWORKS

The idea of using deep learning to help dimensionality reduction tasks is, in itself, not new. Early on, autoencoders have been proposed to reduce the dimensionality n of some data space, sampled by a training set, to a (typically much) lower dimensionality $m \ll n$ [59]. If one sets $m = 2$, this approach basically delivers a projection algorithm from nD to 2D. Refinements of this approach have been proposed in various works, *e.g.* using variational autoencoders [71]. One of the important advantages of deep learning is its *parametric* nature. Namely, the trained network learns (from the provided training set) the structure of the space that these samples imply, and next, during inference, behaves deterministically. That is, given the same (or slightly different) input sample, the network will infer the same (or slightly different) output value. The added-value of this is obvious when computing projections, which should be stable with respect to small changes in the input data, to maintain the user's mental map. A detailed discussion of projections stability has recently been proposed by Vernier *et al.* [157]. Van der Maaten has recognized this earlier, and proposed a modification of the t-SNE method to behave parametrically, using deep learning [81]. Closest to our work, Espadoto *et al.* [41] have recently proposed deep learning to construct *direct* projections based on a (small) training set projected with a user-chosen method such as t-SNE, UMAP, or any other similar algorithm. Our inverse projection method shares many commonalities with [41] – the key difference being that we address the more challenging task of learning a mapping from 2D to nD rather than one from nD to 2D.

In detail, our method (NNinv) works as follows. 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 manifest, *e.g.*, good cluster segregation, distance preservation, or any other known quality metrics [27, 83, 91, 162]. 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 $S_t \subset D$ and its respective projection $P(S_t) \subset P(D)$. We next construct P^{-1} by training a neural network on the training set $T_s = (S_t, P(S_t))$, with S_t selected by random sampling of D. We use the remaining data $D_p = (D \setminus S_t, P(D) \setminus P(S_t))$, 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 S_t. Summarizing, our method has three steps: In step 1, we create the projection $P(S_t)$ of the training samples S_t 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 D_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, neurons 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 [57], and bias elements by a constant value 0.01, which showed good results during testing. We use the Adam [70] 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 Sec. 6.2.1). 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.

6.2 EXPERIMENTS AND RESULTS

We tested our method on the following materials:

Projections: We use for P t-SNE [82] and UMAP [87], which have high-quality and are well known in the dimensionality reduction community [91]. We also tested our method with other projections such as PCA and LAMP. However, given that t-SNE and UMAP score as the best techniques when used as direct projections for computing our decision boundary maps (Chapter 5), we focus next on presenting and discussing the results of NNinv with these two projection methods.

Inverse projections: We compare our method with two alternatives: iLAMP [124] and RBF [5]). 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 intrinsic 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. The *MNIST* dataset [75] has 70K observations of handwritten digits from 0 to 9, rendered as 28 × 28-pixel grayscale images, flattened to 784-element vectors. The *Fashion MNIST* dataset [161] has 70K observations of 10 types of pieces of clothing, rendered as 28x28-pixel grayscale images, flattened to 784-element vectors.

We next discuss our method in terms of scalability (Sec. 6.2.1), quantitative assessment of quality (Sec. 6.2.2), and qualitative assessment of quality (Sec. 6.2.3)

6.2.1 *Scalability in training and inference*

Scalability implies the effort required to *train* our method and, separately, the effort needed to *infer* $P^{-1}(\mathcal{Y})$ as function of the size $|\mathcal{Y}|$ of the dataset $\mathcal{Y}$ to inversely project. Table 5 shows the number of training epochs needed to obtain convergence (defined as in Sec. 6.1) as function of the training set size $|S_t|$, 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 S_t by randomly sampling D (see Sec. 6.1). 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 $|S_t|$ larger than 1K samples.

Figure 6.1 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 S_t, 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 iNN (our method) is almost linear. More importantly, iNN 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 decision

Table 5: Training effort until convergence.

	Average # epochs for each dataset D			
training set size $\|S_t\|$	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 $\|S_t\|$ per D	153.9	136.4	150.6	145.3

boundary maps, presented in Chapter 4 (see [38, 120] and Sec. 6.2.3 next). In such cases, iNN 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 is one of the key reasons why dense maps are built in the first place.

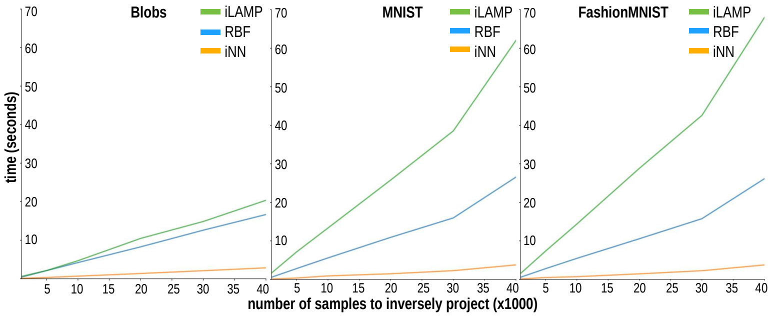


Figure 6.1: Inverse projection speed as function of number of samples.

6.2.2 *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 [83, 140], 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 6.2 shows

MSE for our three datasets, two projections (t-SNE and UMAP), three tested inverse projections (iLAMP, RBF, and iNN). We also consider several training-set sizes $|S_t|$ 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 RBF. *MNIST* and *FashionMNIST* show similar behavior: Our method (iNN) 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.

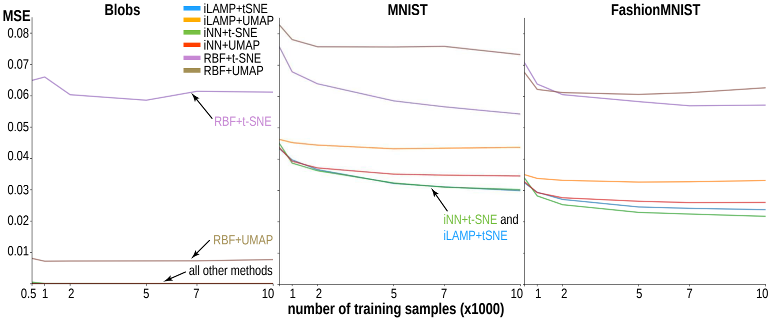


Figure 6.2: Mean square error of inverse projection (lower=better).

6.2.3 *Qualitative Assessment of Quality*

We now show why having a fast and accurate inverse projection is important for our concrete application – understanding the decision zones of classifiers. 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 $P^{-1} \in \{\text{iLAMP}, \text{RBFp}, \text{RBFc}, \text{iNN}\}$, 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 [5]). *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. 6.3). 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 S_t. 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}

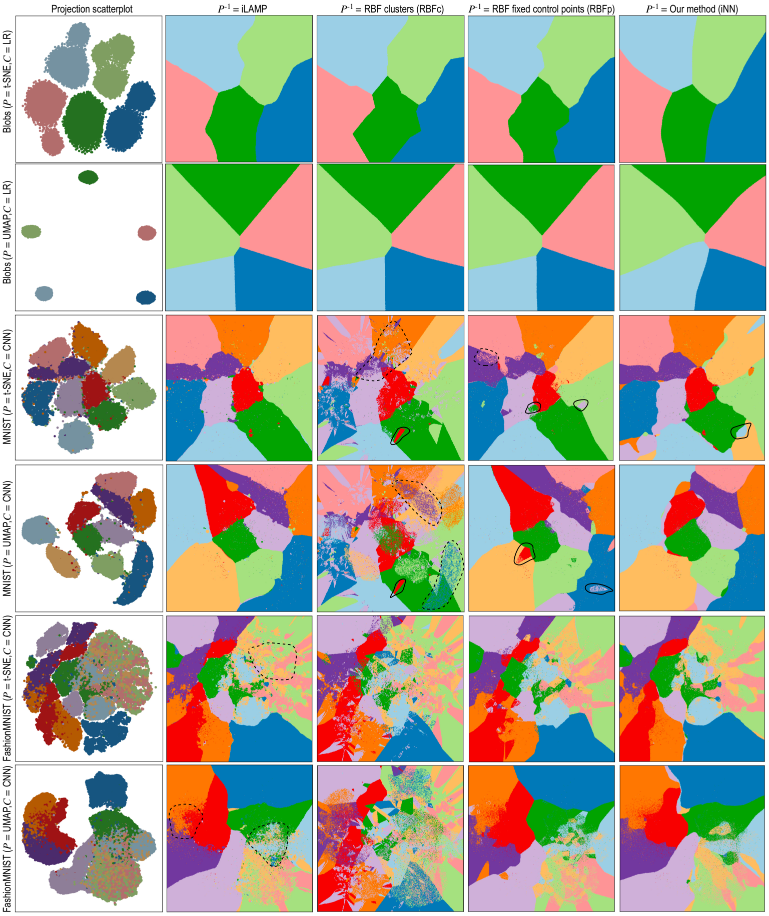


Figure 6.3: Dense maps constructed for combinations of classifiers C, projections P, inverse projections P^{-1}, and datasets. See Sec. 6.2.3.

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 iNN are closest to such linear boundaries, while RBFp and RBFc create more jagged boundaries. This is a first hint that iLAMP and iNN 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 [46]. Hence, we *expect* our dense maps to show this. In Fig. 6.3, we first observe that both iLAMP and iNN 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 iNN. 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 iNN, 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 iNN 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 boundaries (see annotations in figure). Given, again, the mentioned insights on how such zones/boundaries should be [46], we find iNN being better than iLAMP.

6.3 DISCUSSION AND CONCLUSION

We have presented a new method – NNinv – 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), automatic (does not require any user parameters), one to two orders of magnitude faster than existing inverse projection methods (RBF and iLAMP), and simple to implement using existing out-of-the-box deep learning toolkits [23]. 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. As such, we deem NNinv to be the solution of choice for inverse projection when constructing decision boundary maps.

The work on inverse projections is more general, and can be pursued in more directions, than the construction of decision boundary maps. As such, it is interesting to think of the implications and extensions of the NNinv method in a broader sense. First, the design space of NNinv's underlying neural network can be better explored to reach higher accuracy and/or less training effort. For this, the interested researcher could follow the methodology proposed in [39] for the similar task of improving the performance of deep-learned direct projections [41]. 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. This is a particularly interesting topic from a theoretical perspective too, since, to our knowledge, there are no established quality metrics for inverse projections – as opposed to many quality metrics in existence for direct projections. Finally, one can apply our inverse projection to support more applications beyond decision map exploration in machine learning, following the use-cases and examples in [5].

With the above presented comparison of different inverse projection techniques, specifically for the construction of decision boundary maps, we close the loop on covering the two major technical dependencies of the method we proposed back in Chapter 4. In the next chapter we present how decision boundary maps can be *enhanced* to convey more information, thereby making them more effective for analyses related to classifier engineering.

7

VISUAL REFINEMENTS OF DECISION BOUNDARY MAPS

The dense visualization provided by decision boundary maps allows for an explicit representation and visual exploration of a classifer's decision boundaries and decision zones, an improvement in comparison to plain color-coded scatterplots, as discussed in depth in Chapter 4. In Chapters 5 and 6, we have studied the effect of choosing specific direct, respectively inverse, projection techniques on the resulting dense maps, for a variety of synthetic and real-world datasets and classifier techniques.

Overall, the key insight obtained by the above-mentioned experiments is that, for simple datasets (where classes are very well separated in the data space), decision boundary maps constructed by using t-SNE or UMAP (for the direct projection) and our own NNinv (for the inverse projection) match well the expectations we have, such as being smooth for classifiers where we know that this should be the case, such as LR or KNN. However, such ideal conditions are quite far away from real-world cases. Indeed, in practical classification problems, one encounters a far clearer separation of the classes; projections have difficulties in keeping similar samples close to each other in the 2D space; inverse projections suffer from related errors; and more complex classifiers, such as deep learning models, have boundaries which are far more complex in shape.

The experiments discussed in the previous two chapters show that, for the above real-world cases, decision boundary maps suffer from imperfections that manifest themselves as jagged boundaries and numerous small-scale color islands. These can be very problematic for the analysts using such images to understand the behavior of a classifier, as they are not sure whether they are looking at an artifact of the visualization (to be ignored, thus) or an actual problem of the classifier (which should be corrected).

In this chapter, we propose several refinements of the construction of decision boundary maps that aim to alleviate the above issues, as follows. In Section 7.1 we propose to filter out badly projected points, according to a neighborhood based criterion, thereby reducing the amount of island-like noise present in the visualization. In Section 7.2, we enrich the visual encoding of decision boundary maps to display information beyond the sample density and classifier confusion. Specif-

---

This chapter is based on the following publication:
F. C. M. Rodrigues, M. Espadoto, R. H. Jr, and A. Telea. Constructing and visualizing high-quality classifier decision boundary maps. *Information*, 10(9):280–297, 2019

ically, we focus on displaying information showing the distance to decision boundaries. This is motivated by the fact that points close to decision boundaries are more prone to misclassification, thus, such areas are of higher interest to the classifier engineer than "safe" zones located deep inside decision zones. Section 7.3 closes the chapter with a discussion of the proposed refinements, as well as a summary of our contributions in relation to our first research question stated in Chapter 1 and a comparison thereof with other approaches in the literature.

7.1 PROJECTION FILTERING

Throughout this thesis, we explored the construction of decision boundary maps under a very different settings. Even when training completely different classifiers on very different datasets, the resulting DBMs exhibit patterns such as non-smooth decision boundaries and/or small islands in the decision zones (, for example).

In Chapter 5, Sec. 5.3.2, we pointed to the presence of small-scale "islands" on dense maps. These are visible in Fig. 5.4. Further examples of such islands are to be seen in Fig. 4.7. As explained there, such islands are regions of a color (class) completely immersed in a region of a different color (class). As outlined in Sec. 5.3.2, such islands correspond to two different situations:

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

b) the island *may* exist in D, *i.e.*, there exist very similar samples that get assigned different labels. This case can be further split into
    b1) the island *actually* exists in D, *i.e.* 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, *i.e.*, the classifier misclassified points which are similar in the feature space but actually have different labels.

Hence, 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 dimensionality reduction (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 [8, 85, 86, 91]. In particular, the problem case (a) is created by so-called *false neighbors* [85], *i.e.*, points which are far away in the data space (thus, likely, have different labels) but project close to each other (thus, create islands). Note that the other type of projection artifact discussed in [85], *missing neighbors*, *i.e.* points which are close in data space but get projected far away in visual space, is also very likely to create islands. Indeed, a missing neighbor *must* be projected somewhere in the 2D space, so it will become implicitly a false neighbor. Interestingly, this quite obvious relationship between false and missing neighbors has not been further discussed in [85]. The same limitations (concerning projection errors) are shared by the inverse projection P^{-1} [5, 42, 124].

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 [91]; neighborhood hits [66]; false neighbors, missing neighbors [85]; and the projection precision score [126]. Given our goals of characterizing how well a nD *compact* neighborhood maps to a similarly-compact 2D neighborhood, we use here the Jaccard set-distance [86] between the k-nearest neighbors $\nu_k^2(i)$ of a point in the $2D$ projection and its neighbors $\nu_k^n(i)$ in nD, given by

$$JD_k(i) = \frac{|\nu_k^2(i) \cap \nu_k^n(i)|}{|\nu_k^2(i) \cup \nu_k^n(i)|}, \tag{7.1}$$

where $|\cdot|$ denotes set size.

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 (Eqn. 7.1), we next filter out from the projection low-ranked points and construct the dense map from the remaining points as usual. Setting an absolute removal threshold is however hard, and moreover depends on the neighborhood size k. To explain this, Fig. 7.1 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,

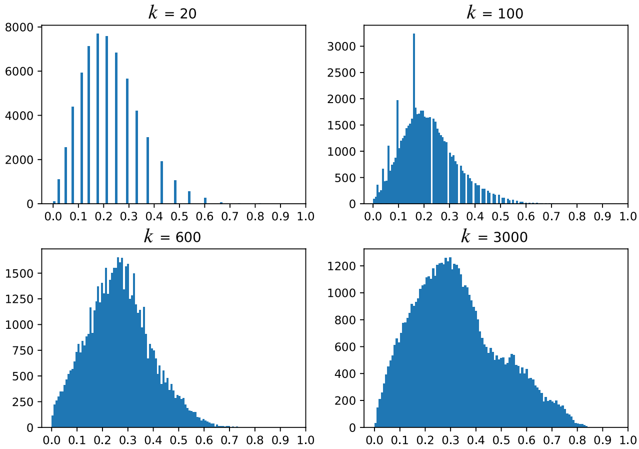
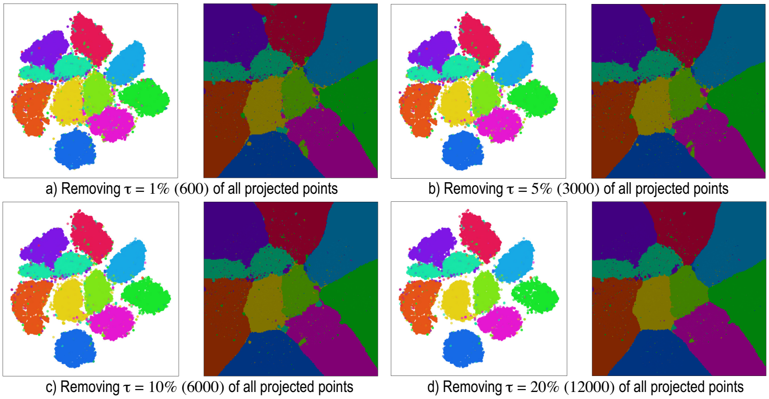


Figure 7.1: Histogram of JD_k rank for varying values of k for MNIST dataset, t-SNE projection.

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 7.1 shows a second, equally important, aspect, namely that the signal JD_k has a *discrete* nature. Indeed, for a given k, Eqn. 7.1 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 *e.g.* the vertical axes of the images in Fig. 7.1 for low *vs* 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 7.2 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 island artifacts and keeping enough data to have an


a) Removing $\tau = 1\%$ (600) of all projected points

b) Removing $\tau = 5\%$ (3000) of all projected points

c) Removing $\tau = 10\%$ (6000) of all projected points

d) Removing $\tau = 20\%$ (12000) of all projected points

Figure 7.2: 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.

insightful dense map. This is the setting used next in all images in this chapter.

7.2 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 of data in D such problems occur, and next alleviate this by *e.g.* adding more training samples of that kind (class).

To analyze this further, let us introduce some notations. Let $Z \subset \mathbb{R}^n$ be a decision zone in the data space, and let $\partial DZ \subset \mathbb{R}^n$ be the boundary of this zone. The decision maps, constructed as explained so far, do not show the distance $d_{nD}(\mathbf{x})$ from a sample $\mathbf{x} \in D$ to its closest decision boundary ∂DZ in the nD space. Rather, the maps show 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

$$d_{2D}(\mathbf{y}) = \min_{\mathbf{y}' \mid f(\mathbf{y}) \neq f(\mathbf{y}')} \|\mathbf{y} - \mathbf{y}'\| \tag{7.2}$$

can thus be seen as a *projection*[1] of the actual nD distance $d_{nD}(\mathbf{x})$ we are interested in. The two distances are not the same, nor even linearly related, given the local compression and stretching caused when mapping the nD space to 2D by nonlinear projections such as t-SNE or UMAP [8, 91]. Note that Eqn. 7.2 is nothing but the so-called distance transform [43] of the set of pixels that constitute the projections of the decision zone boundaries ∂DZ.

An *exact* computation of d_{nD} is impossible in general, since we do not have an analytic description of ∂DZ for typical classifiers. Simply put, we do not know where ∂DZ are located in data space – if we knew this, our entire endeavor would have been solved from the beginning. Hence, we next propose two classifier-independent heuristics to estimate d_{nD} (Secs. 7.2.1 and 7.2.2) as well as a third, more exact, method, and better suited for neural network classifiers, based on adversarial examples (Sec. 7.2.3).

Figure 7.3 compares the 2D distance-to-boundary d_{2D} (computed by Eqn. 7.2, implemented using the fast distance transform method in [20]), 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 Fig. 7.3 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 logistic regression 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 in Chapter 6 to test the quality of the NNInv inverse projection.

In Figure 7.3, 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 Fig. 7.3b, *i.e.*, are at the same distance-to-boundary (d_{2D}) in 2D, can have quite different colors in Figs. 7.3c,d, *i.e.*, 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 sample is.

7.2.1 *Image-based Distance Estimation*

For every pixel $\mathbf{q}$ in the dense map, we find the closest pixel $\mathbf{r}$ having a different label (Fig. 7.4a). Let Q and R 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

---

[1] We use projection here in the weak sense of the word. Indeed, we cannot formally say that $d_{2D}(\mathbf{y}) = P(d_{nD}(\mathbf{x}))$, since a projection P maps points, not distances.

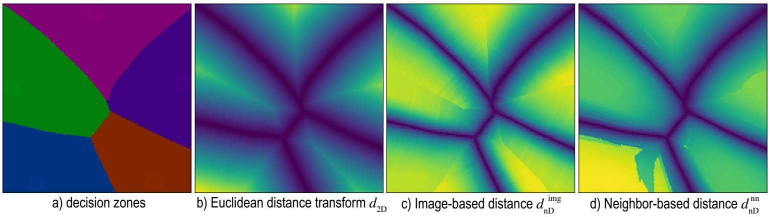


Figure 7.3: Dense map (a) and various distance-to-boundary maps (b-d) for Blobs dataset, computed using UMAP for P and NNInv for P^{-1}.

and R 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, *i.e.*, 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

$$d_{nD}^{img}(\mathbf{q}) = \frac{1}{|Q||R|} \sum_{\mathbf{x}_Q \in Q, \mathbf{x}_R \in R} \|\mathbf{x}_Q - \mathbf{x}_{QR}\|. \tag{7.3}$$

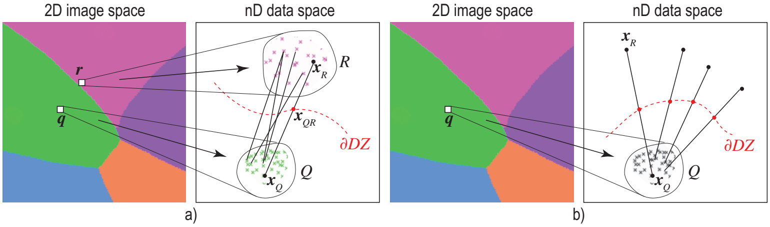


Figure 7.4: Estimation of distance-to-boundary d_{nD}^{img} (a) and d_{nD}^{nn} (b). See Secs. 7.2.1 and 7.2.2.

Although Eqn. 7.3 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 $P^{-1}(\mathbf{r})$.

7.2.2 *Nearest-neighbor Based Distance Estimation*

We can improve upon the dense map-based heuristic presented in Sec. 7.2.1 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 (Fig. 7.4b). 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

$$d_{nD}^{nn}(\mathbf{q}) = \frac{1}{|Q|} \sum_{\mathbf{x}_Q \in Q, \mathbf{x}_R = \underset{\mathbf{x} \notin Q | f(\mathbf{x}) \neq f(\mathbf{x}_Q)}{\arg\min} \|\mathbf{x} - \mathbf{x}_Q\|} \|\mathbf{x}_Q - \mathbf{x}_{QR}\|. \tag{7.4}$$

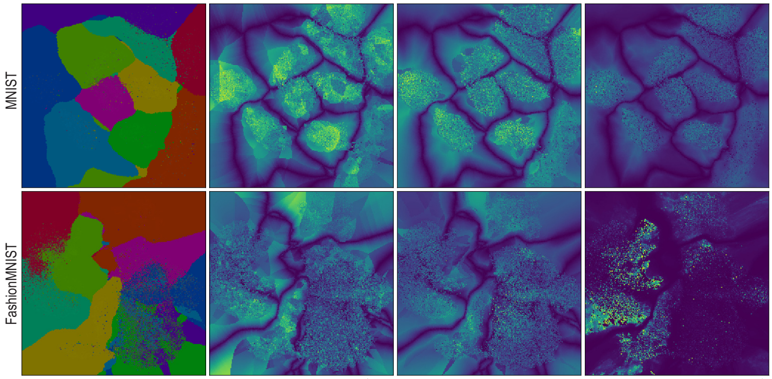
Estimating d_{nD} this way is more accurate than using Eqn. 7.3 since we do not rely on computing $\mathbf{x}_R$ using the possibly inaccurate dense map, but directly use the nD points S. We implement Eqn. 7.3 by searching for nearest neighbors in nD space using the *kd*-tree spatial search structure provided by *scikit-learn* [104].

7.2.3 *Adversarial Based Distance Estimation*

The third proposed heuristic is based on adversarial examples [51, 143]. 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 $\mathbf{x} + \epsilon$, *i.e.*, 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

$$d_{nD}^{adv}(\mathbf{q}) = \frac{1}{|Q|} \sum_{\mathbf{x}_Q \in Q} \min_{f(\mathbf{x}_Q) \neq f(\mathbf{x}_Q + \epsilon)} \|\epsilon\|. \tag{7.5}$$

Compared to the distance-to-boundary heuristics given by Eqns. 7.3 and 7.4, Equation 7.5 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 Eqn. 7.4 and 30 times more than evaluating Eqn. 7.4. 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 [51].



a) decision zones b) image-based distance $d_{\mathrm{nD}}^{\mathrm{img}}$ c) neighbor-based distance $d_{\mathrm{nD}}^{\mathrm{nn}}$ d) adversarial-based distance $d_{\mathrm{nD}}^{\mathrm{adv}}$

Figure 7.5: 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.

Figure 7.5a 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 Eqns. 7.3, 7.4, and 7.5, respectively, visualized using the same luminance colormap as in Fig. 7.3. Several observations follow. First, we see that the nD distances d_{nD} roughly follow the patterns of the 2D Euclidean distances d_{2D}, *i.e.*, 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 (Sec. 7.2.1). 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.

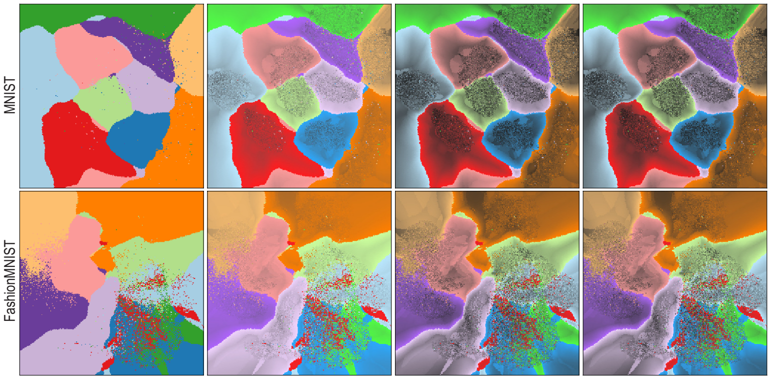
7.2.4 *Visualizing Boundary Proximities*

Visualizing the raw distance d_{nD} by direct luminance coding (Fig. 7.5) 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 in-correctly, 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

$$V(\mathbf{q}) = 0.1 + 0.9(1 - d_{nD}^{nn}(\mathbf{q})/d_{max})^{k_1} \tag{7.6}$$

$$S(\mathbf{q}) = S_{base}(1.0 - d_{nD}^{nn}(\mathbf{q})/d_{max})^{k_2} \tag{7.7}$$

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 7.6 shows the effect of k_1 and k_2 for the MNIST and FashionMNIST datasets. Compared to showing only the decision-zone information (Fig. 7.6a), 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.



a) decision zones b) blended map $k_1 = 0.5, k_2 = 0.3$ c) blended map $k_1 = 1.5, k_2 = 0.7$ d) blended map $k_1 = 2, k_2 = 0.9$

Figure 7.6: (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 7.6 is to be interpreted as follows: Dark areas indicate data samples deep inside decision zones, *i.e.*, 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 nD 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 nD 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.

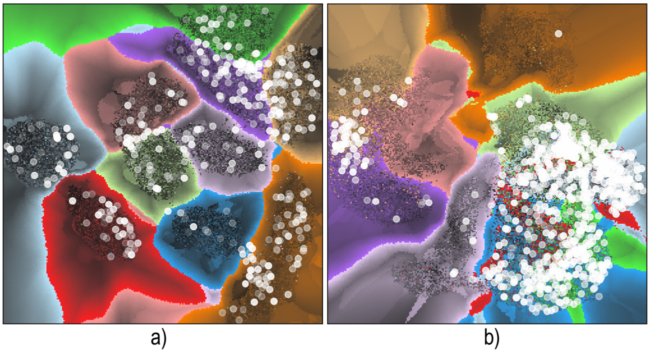


Figure 7.7: Misclassifications with opacity coding distance-to-boundary for the (a) MNIST and (b) FashionMNIST datasets.

Figure 7.7 shows a different use-case for distance maps. Atop of the distance maps shown in Fig. 7.6 ($k_1 = 2, k_2 = 0.9$), 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 *e.g.* by data annotation or augmentation.

7.2.4.1 *Enridged Distance Maps*

Figure 7.6 encodes distance-to-boundary by luminance and saturation, which are good visual variables for *ordinal* tasks, *e.g.*, estimating which points are closer or farther from decision boundaries. However, this encoding is less suitable for *quantitative* tasks, *e.g.*, 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 [159]. 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 Eqns. 7.6 and 7.7. Note that the transfer function f is only *piece wise* continuous and, as shown in [159], 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 Secs. 7.2.1 - 7.2.3), filtering is required. Besides filtering, a second difference between our approach and the original technique [159] is that we visualize directly the *distance*, whereas [159] 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 7.8 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.

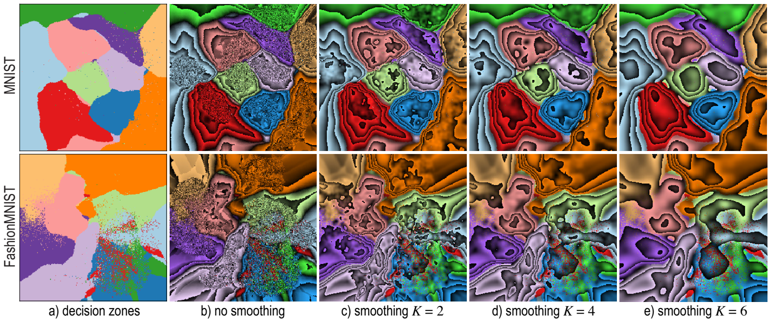


Figure 7.8: Enridged distance maps for MNIST (top row) and FashionMNIST (bottom row) datasets. Images (b-e) show the progressive noise-smoothing effect of the filter radius K.

7.3 DISCUSSION

The refinements presented in this chapter – namely, the filtering of islands created by projection errors; and the visual depiction of distance-to-boundary over the decision zones conclude our proposed designs for visualizing decision boundary maps, and therefore our technical contributions in answering the first research question posed in Chapter 4, which we repeat below

> *How can we depict the decision boundaries of a classifier and use these to understand its operation and performance?*

While a detailed discussion of our entire work is given in Chapter 9, we present next a summary of our technical contribution, highlighting its key assets.

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

Technical foundations: Our method is based on the application of direct and inverse projections. We have explored both spaces of techniques in Chapter 5 and 6, respectively, collecting insights on which techniques work best for the construction of decision boundary maps.

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 work 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 7.1). Also, showing the distance-to-boundary (Section 7.2) can highlight the presence of remaining errors.

Novelty: While dense maps have been used earlier in high-dimensional data visualization to analyze projection quality [8, 85], 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 Schulz *et al.* [127] 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 theirs exist, as follows:

- *Computation of inverse projection P^{-1}:* In their method, 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 6, 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:* For [127], 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 7.1) achieves roughly the same effect as discriminative DR but at a lower computational cost and with a very simple implementation;

- *Distance to boundary:* In [127], this quantity, which is next essential for creating dense decision boundary maps, is assumed to be given by the projection algorithm P. Quoting from [127]: "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 7.2).

- *Computational scalability:* Schulz *et al.* [127] 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* above).

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 [13]. 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. We explore this idea in the next chapter.

8

END TO END EVALUATION

Over the previous chapters, decision boundary maps (DBMs) were presented as a tool to provide insights in the way classifiers partition their data space. The focus of our work so far has been in *constructing* DBMs that represent the actual decision boundaries and decision zones as accurately as possible, subject to the inherent limitations posed by direct and inverse projections, as well as the finite resolution of the images used to synthesize the DBMs. By this, we have attempted to answer our first research question outlined in Chapter 1.

In this chapter, we turn our attention to the actual *usage* of DBMs to assist classifier engineering. Specifically, we explore how DBMs can be used to improve the accuracy of a given classifier by analyzing a Semi-Supervised Learning (SSL) with a *human-in-the-loop* scenario. For this, we developed an interactive visual analytics tool, based on DBMs, that allows the classifier engineer to assign labels to a set of unlabeled samples. This is done based on the visual cues jointly presented by projected samples and DBMs constructed using the visual refinements from Chapter 7. We next used this tool to conduct several experiments involving different datasets and classifier models. These experiments aim to evaluate the usefulness DBMs: If a human user can correctly assign labels to given data points, it means that the visual hints given by the tool were sufficiently informative and accurately reflect data behavior in original data space. At a higher level, if by doing this one can improve the performance of a classifier, it means that DBMs have demonstrable added value in classifier engineering.

The structure of this chapter is as follows. Section 8.1 introduces SSL and the added value of the human-in-the-loop model. Section 8.2 presents a visual analytics tool that we have designed to assist SSL, in particular label propagation, in which DBMs play a key role. Sections 8.3 and details several experiments that we have conducted to assess the efficiency and effectiveness of our proposed visual tool. Section 8.4 details the results of these experiments. Section 8.5 discusses our end-to-end proposal.

8.1 SEMI SUPERVISED LEARNING

A SSL setting commonly supposes that labeled data is scarce. In most real-world problems, data is labeled by domain experts, thus creating a dataset of annotated samples can be a costly activity. For many problems it is usual that abundant unlabeled data is available while little labeled data is at disposal [136]. SSL approaches seek for ways to use information present in the unlabeled dataset to build better machine learning models than ones that could be inferred using only the little amount of available labeled data.

Common methods for SSL involve heuristics to guess labels for a unlabeled dataset U. The most naive approach consists of training a classifier f on the set of labeled samples L, using it to *predict* the labels of data in the unlabeled set and training f again with the induced labels $L \cup f(U)$. It is expected that iterative repetitions of those steps lead to a robust model that learned from features present in U [136].

A second class of approaches *propagates* the labels from samples in L to the samples in U by using information present in the data, such as exploring neighborhood information [158], geometric information [12], constructing similarity graphs from the data samples [6, 79], and metric learning [52]. Detailing all these methods is beyond the scope of this thesis. The interested reader can refer for this goal to various surveys [108, 111, 169].

The key advantage of automatic label propagation techniques is that they require little or no effort from the user to be deployed in order to generate a rich enough labeled training set for the subsequent training of a classifier. Such techniques have been demonstrated to be effective in many contexts where small labeled training sets were available. However, from a conceptual viewpoint, such techniques make the same (strong) assumption that classifiers do – namely, that they can *extrapolate* information from a (small) given labeled dataset to points outside it. One difference, in this context, between classifiers and label propagation techniques is that a classifier does not know, at training time, which are the points (test set or validation set) on which it will be used next; in contrast, label propagators do typically know this as they have the locations of the unlabeled data points that they need to propagate to. Still, as said above, the automatic nature of both classifiers and label propagators assumes that such techniques can correctly "guess" the label of a data point based on the structure of the data around it.

A related, but different, approach is taken by combining visual analytics (VA) with label propagation. The key idea behind this is that a human can spot relevant structure in a high-dimensional dataset by visualizing a (low dimensional) representation thereof, and, based on this structure, can propagate labels better than an automatic method can. This was demonstrated recently by Benato *et al.* [13]. Their work projects a high-dimensional dataset to 2D using t-SNE and colors the resulting scatterplot by the labels of the (few) available samples. Next, the user infers how to propagate labels based on the structure of data present in the projection. For instance, if one sees a compact cluster of unlabeled points in which a few labeled points exist with label c, then one can decide (based on additional information, *e.g.*, exploring the actual samples by means of image tooltips) to propagate c to the entire cluster. The authors have compared this semi-supervised labeling strategy based on the human-in-the-loop with automatic label propagation techniques such as Laplacian Support Vector Machines (LapSVM [12, 136]) and Optimum Path Forest (OPF-Semi [6]). They showed that the VA-based approach can achieve superior performance – both in the label propagation accuracy and in the accuracy of a classifier trained with the propagated labels – as compared to automatic label propagation techniques.

The above is a very interesting result, as it points to the fact that a human user can discover more information (for label propagation, that is) in a two-dimensional, and necessarily imperfect, projection, than an automatic tool can do even when having access to the full high-dimensional data. While Benato *et al.* do not further speculate or analyze why this is so, we believe that this has to do with the fact that the human user literally "sees" more complex data patterns, appearing at different scales, in the projection, than a typical automatic label propagator can do. For instance, they report that the user will propagate labels with a high confidence when they see a "clearly separated" cluster of observations which is also "far away" from other similar clusters; in contrast, the user will be reluctant to propagate labels in areas in the projection where one sees a mix of different labels and/or which are close to the fuzzy separation frontier of two clusters. Given our understanding of how automatic label propagation techniques work, such patterns are not detected or used for propagation by such techniques, which typically work in a more local fashion.

Our work in this chapter parallels (and extends) the work of Benato *et al.*. Specifically, while Benato *et al.* used only a color-coded projection scatterplot to perform the manual propagation, we use, for the same task, the same projection scatterplots overlaid over the DBMs. We measure the added value of the visual hints provided by DBMs by several experiments along the same lines as Benato *et al.*.

8.2 VISUAL ANALYTICS FOR SEMI SUPERVISED LEARNING

Our proposal for a VA approach in a SSL setting consists in a visual tool that conveys to the user information about a dataset and classifier through the rendering of DBMs and color-coded projections. To explain this tool, we introduce first some notations. Let $\mathcal{D} = (X_{nD}, Y)$ be a dataset consisting of a number of n-dimensional samples X_{nD} and their corresponding categorical labels Y. Let X_{2D} be the projection of the dataset X_{nD} computed by any suitable dimensionality reduction technique, *e.g.*, t-SNE. Let f be the classifier chosen by the user to engineer, *e.g.*, SVM.

Our tool creates and displays two main elements – the projection X_{2D} color coded by class labels and, optionally, misclassification information (when used in testing mode), and the DBM computed from $\mathcal{D}$ and the trained classifier f, using the techniques presented in Chapter 7.

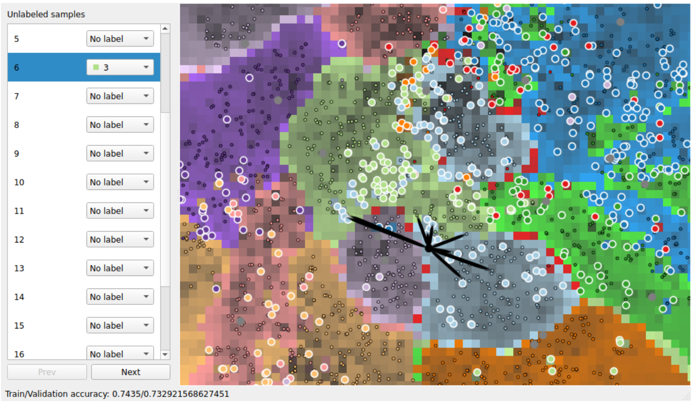
Further on, the tool provides details-on-demand on specific samples by mechanisms such as brushing and selection.

The visual analytics workflow starts by loading the original dataset $\mathcal{D}$ and selecting a classifier f of interest. Next, the projection X_{2D} is computed, using either t-SNE or UMAP. The user then defines how the data is going to be split into three sets: labeled (L), unlabeled (U), and validation (V). The classifier is trained using the samples in L, while the labels of the samples in U are hidden from the user. Classifier performance is evaluated using all samples in L, which yields a training error; and also using all samples in V, which yields a validation error. At this point, the user can decide how to continue: If the training and validation errors are low enough for the problem at hand, one concludes that the training was successfully completed, and the process stops. If not, the visual analytics process for improving the classifier starts, as described below.

Using the loaded data and trained classifier, a DBM is computed and rendered. On top of the DBM, the set of projected points X_{2D} is shown. In this set, misclassifications are highlighted by a white border. Next, the tool renders in dark grey $N = 20$ unlabeled points at a time, in order they appear in the unlabeled set. We next call these points the working set. Keyboard shortcuts are provided allowing the user to show additional working sets of N points, and to go forward and backward through the list of unlabeled samples U. The user can next visually inspect the working set points in terms of their neighbors in the projection and their location with respect to the decision boundaries. Since, as explained already at several moments, the projection technique P cannot preserve all neighbors, we provide a tool to show the true nD neighbors $\nu(\mathbf{x})$ of each point $\mathbf{x}$ in the working set by linking the projection $P(\mathbf{x})$ of $\mathbf{x}$ with lines to all points $\{P(\mathbf{y})|\mathbf{y} \in \nu(\mathbf{x})\}$. This effectively shows the missing neighbors of $\mathbf{x}$ in the projection. A depiciton of the Graphical User Interface of the tool, with all the feature just described, used to label samples is shown in Fig. 8.1.

Based on all above visual hints, the user can next decide how to propagate labels to points in the working set. The overall idea here is similar to the proposal of Benato *et al.* [13]. That is, one would typically propagate a label (of a point $\mathbf{x} \in L$) to points in U which project close to $P(\mathbf{x})$ and are not in or close to confusion zones of the classifier, as depicted by the underlying DBM. They key difference between our work and Benato *et al.* is that we provide more hints (in terms of the DBMs and true neighbors) to the user besides the color-coded scatterplot. If one does not find enough suitable points to propagate to, one can advance to the next working set in U, as described above. We chose this working set design rather than allowing the user to see all the unlabeled points U at a time so as to limit overplotting and visual and interaction complexity.

Once the user assigns a label to a point in U, the point is moved out of U in a set of manually labeled points L_m. Initially, L_m is empty. Next,


Unlabeled samples

5	No label
6	🟩 3
7	No label
8	No label
9	No label
10	No label
11	No label
12	No label
13	No label
14	No label
15	No label
16	No label

Train/Validation accuracy: 0.7435/0.732921568627451

Figure 8.1: Screen capture of the visual analytics tool designed to assist manual labeling by using DBMs. A projection is shown on top of the decision map, with samples marked as labeled, unlabeled, or wrongly labeled (by a trained classifier). Users can analyze the positions of these samples with respect to the underlying DBM to decide which of the unlabeled ones they next want to label, and which labels to assign to these.The left widget lists the unlabeled samples and allows manually selecting labels from these from the drop-down menus. For a selected unlabeled sample, the main window also shows its five closest nD neighbors by black lines.

the user trains the same classifier on the enriched label set $L \cup L_m$ and re-evaluates the obtained performance. Three cases can occur concerning this performance after retraining:

- *Performance improves:* In this case, the user likely keeps the labels added in the last iteration of constructing L_m. If more training effort (time) is available and the performance is still not the desired one, the user continues manual labeling with the next working set;
- *Performance degrades:* In this case, the user likely undoes (removes) the labels added in the last iteration of constructing L_m;
- *Performance stays constant:* In this case, one typically keeps the labels added in the last iteration of constructing L_m, even if performance slightly drops. Indeed, it might be useful to keep the label assignment, as this provides more information to the classifier to learn from.

After each iteration, the user can decide whether to continue or not the labeling process, based on the classifier performance obtained so far and the amount of effort one wishes to spend in manual labeling. If one decides to continue, the next working set is examined. Since the classifier is retrained after each iteration, a new DBM is computed for the new classifier and shown to the user. This way, one effectively sees how the decision boundaries *move* due to the assignment (or removal) of labeled points, thereby conveying an idea of how the learning process actually uses the training set. As a side note, we completely retrain the classifier after each batch of label assignment is finished for implementation simplicity. Alternatively, one could apply *online training*, *i.e.* update the classifier using the new labeled samples in the current iteration.

8.3 MANUAL LABELING EXPERIMENTS

We designed five experiments of different dataset-classifier combinations in increasing order of complexity to assess the quality of manual sample annotation. The scenarios go from simple linear classifiers to convolutional neural networks (Sec. 8.3.1); and from synthetic two-class dataset to ten-class natural image data (Sec. 8.3.2).

To assess the quality of the manually assigned labels, we compare these with an automatic label (AL) propagation technique, at two levels of detail:

- *Full automation:* We first compare manual labeling against AL'ing the entire U. This scenario answers the question: What is more effective – to let AL handle all samples or to invest manual effort?
- *Same effort:* We next compare manual labeling with k randomly drawn samples from AL, where k is the number of manually assigned labels. Note that in most cases $k \ll |U|$, as the user typically labels only a small fraction of all available unlabeled points, either because manual labeling starts being time-consuming after a while, or because visual hints are not informative enough to allow one to confidentially label certain points. This scenario answers the question: Given the same number of labels k, who can produce better ones, AL or the user?

As AL method, we use a standard graph based label propagation method [31], whose implementation is readily available in *scikit-learn*. If desired, other AL methods can be immediately used, *e.g.*, LapSVM [12, 136]) or OPF-Semi [6].

8.3.1 *Classifiers description*

We used three types of classifiers in the experiments: a Logistic Regression (LR) model, a shallow Multi-Layer Perceptron (MLP) and a small Convolutional Neural Network (CNN). A detailed description of each of those classifiers is presented next.

Logistic Regression: A linear classifier is arguably one of the simplest forms of machine learning discriminant. Intuitively, the rigid shape of the decision boundaries induced by such classifier indicates that supplying new labeled points might not be helpful to change them. We used the *scikit-learn* implementation of this classifier with default parameters.

Multi-Layer Perceptron: Nonlinear classifiers, in particular neural networks, allow for the formation of complex-shaped decision boundaries in data space. Thus, it is reasonable to expect that labeling new samples could cause modifications *locally*, allowing a user to effectively influence (by manual labeling) the shape of the partitions induced by the classifier. In the experiments presented next, we used MLP consisting of three hidden layers of sizes 32, 32, and 16 units respectively, trained with early stopping and the *Adam* optimizer, as provided by the Python package *scikit-learn*.

Convolutional Neural Network: CNNs are the most used model for computer vision tasks, thus experimenting with such model is of practical relevance. We used the *PyTorch* library to create a small CNN with the following configuration: a convolutional layer consisting of 6 filters of sizes 5×5, max pooling layer of size 2×2, a second convolutional layer consisting of 16 filters of sizes 5×5, another max pooling layer of size 2×2, followed by two fully-connected layers of sizes equal to 60 and 30 units respectively. ReLU activation function was used for every layer. We use this CNN on two datasets, one consisting of four different classes and another that consists of ten classes. Hence, the last (output) layer of the CNN contains 4, respectively 10 units, depending on the dataset.

8.3.2 *Datasets description*

We used five different datasets in the experiments of manual sample annotation, as follows.

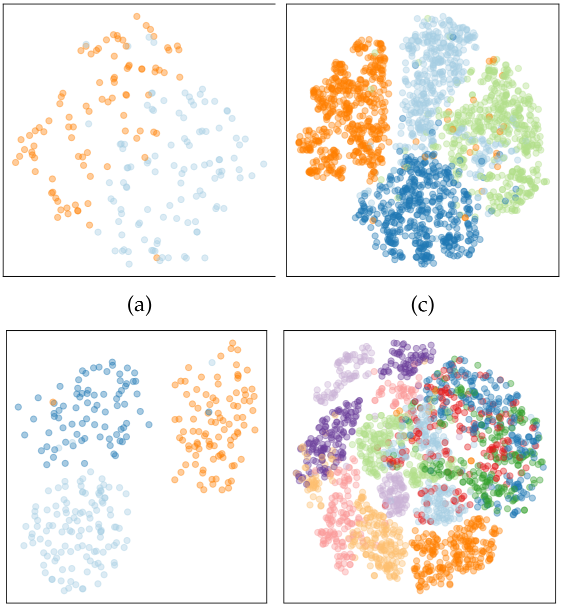
Two-class synthetic (syn2a): Generated from two Gaussian clusters, this dataset is composed of 600 samples of dimensionality 10, split into labeled, unlabeled and validation sets as follows: $|L| = 200$, $|U| = 200$, $|V| = 200$. A t-SNE projection of this dataset in shown in Figure 8.2(a) and was used as input to the visual tool.

Two-class synthetic (syn2b): This dataset is very similar to the previous one. It is generated with the same set of parameters, but consists of 6000 samples. This dataset was split into $|L| = 200$, $|U| = 1200$, $|V| = 3600$ by randomly selecting the respective number of points from the dataset. We use the same procedure to split the other datasets as well.

Three-class synthetic (syn3): This dataset was generated from three Gaussian clusters in 10 dimensions and consists of 6000 samples. We split this dataset into $|L| = 200$, $|U| = 1000$, $|V| = 3800$. A UMAP projection of this dataset is shown in Figure 8.2(b).

Four-class image (fm4): This dataset is a subset of FashionMNIST and contains only the first four classes (T-shirt/top, Trouser, Pullover and Dress). It contains 24000 images, 6000 of each class, of size 28×28 split into $|L| = 2000$, $|U| = 1000$, $|V| = 21000$. A t-SNE projection of this dataset is shown in Figure 8.2(c).

Ten-class image (fm10): The last dataset used in our experiments consists of all the ten-classes in FashionMNIST, split similarly to *fm4* but with a much larger validation set ($|L| = 2000$, $|U| = 1000$ and $|V| = 51000$). We removed 6000 of the worse-projected points according to the Jaccard Distance, following the ideas presented in Chapter 7. A t-SNE projection of this dataset is shown in Figure 8.2(d).



(a) (c)

(b) (d)

Figure 8.2: Projections for the datasets used during the experiments. (a) 2-class synthetic, (b) 3-class synthetic, (c) 4-class FashionMNIST subset, (d) 10-class FashionMNIST. In all cases, we only show the points L for which we have label information. See Sec. 8.3.2.

8.3.3 *Experimental set-up*

Combining all the three classifiers (Sec. 8.3.1) with all five datasets (Sec. 8.3.2) would create fifteen potential experiments to execute. While this is certainly doable, we argue that it would not be optimal. For instance, it does not make much sense to use LR (a very simple and not that powerful classifier) for *fm10*, which is a very challenging dataset to classify. Hence, from this total of fifteen possibilities, we selected five classifier-dataset combinations to experiment with. These are listed under the names (I - V) in the leftmost column of Tab. 6. We chose these five combinations so as to match the perceived classification challenge (implied by the complexity of a dataset) to the classifier power and flexibility, and also to cover a wide range of possibilities.

Table 6: Initial accuracies for each experiment.

Experiment	Baseline Accuracy	
	Train (T)	Validation (V)
(I) **LR** and **syn2a**	87.5%	84.0%
(II) **MLP** and **syn2b**	64.0%	58.9%
(III) **MLP** and **syn3**	76.0%	68.8%
(IV) **CNN** and **fm4**	86.7%	85.3%
(V) **CNN** and **fm10**	74.6%	73.0%

Figure 8.2 shows the initial projection of the labeled samples L for the datasets *syn2*, *syn3*, *fm4*, and *fm10*. The projected points are color coded by their respective trained classifier. That is, each point is assigned a categorical color depending on which class the classifier assigned to it. Note that, in case of training errors, this color will not be the same as the sample's true label from L. This view reveals the classifier "sees" the training set.

Table 6 presents the baseline accuracies of each experiment. By baseline, we mean here the accuracies computed when training each classifier on the set L of available labels for the respective datasets (Sec. 8.3.2). For Experiments I, IV and V, train and validation accuracies are quite close, a sign that there was no overfitting. For Experiments II and III, a big discrepancy between train and validation errors hints that not enough data was available for training, that is, the classifier has too many parameters to adjust, thus causing overfitting. Apart from that, the overall (training and validation) accuracies are quite low, which indicates that these classifiers could be improved. We will aim to do precisely this using labels created manually using the visual analytics workflow outlined in Sec. 8.2.

8.4 MANUAL LABELING RESULTS

We next present the use of our visual analytics tool (Sec. 8.2) to create manual labels for the five experiments described in Sec. 8.3. For each experiment, we used three iterations of the labeling workflow. An iteration consists of examining the DBM of the currently-trained classifier, and labeling a small batch of typically 5 to10 samples, after which classifier retraining and recomputation of the DBMs is done. We believe that this set-up reflects quite well what a typical user would do when using our tool in practice. Indeed, having larger batches would imply that one needs to put more labeling effort before actually seeing what this effort leads to (classifier accuracy increase, stagnation, or decrease). If the accuracy decreases, the labeling needs to be undone (see Sec. 8.2), which means more effort is lost than if a smaller batch was used. Conversely, having smaller batches would imply that the classifier needs to be retrained and the DBMs need to be recomputed more frequently, which is computationally intensive. Moreover, using smaller batches may mislead the user – one should not, after all, take decisions based on how a single sample affects a classifier or its DBM.

We next present several snapshots of the DBMs computed during this iterative labeling process and discuss them, as follows.

Initial situation: Figure 8.3 shows the initial DBMs for all five experiments. Projected points are shown on top of the decision boundary maps. Misclassified points are highlighted by a white outline. This information is important and serves as guidance to manual labeling. For instance, the user can decide to manually label samples that are close to (groups of) misclassified samples, so as to "push" the decision boundaries in the right direction. Apart from this, the images in Fig. 8.3 show several other insights. For Experiment I, we see that most misclassifications are quite close to the left decision boundary which separates the orange from the light blue decision zone. Since most of these misclassifications are blue, this likely means that the boundary is drawn too far to the left – that is, points that should have been orange are actually classified as blue. Hence, manual labeling should aim at pushing the decision boundary to the right, deeper into the blue zone. For Experiment II we see a different situation. Here, misclassifications are spread deeper into the decision zones. Solving these by modifying the shape or location of the decision boundary is arguably more complex. Experiment III shows a situation roughly similar to the one for Experiment I. Here, we see that most misclassifications are either orange points located inside the top dark-blue decision zone (these points should have been classified as blue), or blue points located in the large light-blue decision zone to the left (these points should have been classified as orange). Hence, the orange decision zone should extend deeper inside the light-blue zone and the dark-blue zone should be smaller. The DBMs for Experiments IV and V are more complex and harder to interpret due to the larger number of misclassifications and also the larger number of classes.

Iterative labeling: Figures 8.4 and 8.5 show how the decision maps change after three iterations of labeling for each of the five experiments. For each experiments, the figures show six images (a-e), structured as follows: The three images on the top row (a,c,e) show the points labeled by the user, in each of the three consecutive iterations) as full white disks, rendered atop the DBMs visible at that moment. The corresponding images in the bottom row (b,d,f) show the *effect* of the respective labeling iteration, *i.e.*, how the DBMs change and which are the misclassifications that the classifier, trained with the labeled points shown in the top row, produces. That i: The user started the process seeing the DBM in image (a); after adding some labels and retraining, she could observe the effect in (b). From this visual insight, the user decided to add more labels, as shown in (c), leading to the result in (d), and so on.

These image sequences offer several insights, as follows. First, in general, we see that the *changes* in the DBMs, due to the manually added labels, are quite small and local. This is not unexpected, since we add only a few tens of labels manually to training sets that contain hundreds up to thousands of labels. So, small changes to the training sets imply small changes to the behavior of the respective classifiers and their DBMs.

More specifically, we see that, for the easier problems, manual labeling has a visible and positive impact. For instance, in Experiment I(b) (Fig. 8.4), we notice a small cluster of misclassified points in the bottom-left area of the image. The user notices this, and adds a few labels manually in this area (white dots in Experiment I(c) (Fig. 8.4). The classifier is retrained using this information, and the number of misclassifications in this area drops (Experiment I(d), Fig. 8.4). In the same time, we see that this adjustment of the decision boundaries is quite *local*: While the classifier improves in this area, it does not get worse far away from it – the number and positions of misclassified points far away from this area stays the same. The user next concentrates on the few misclassifications located deeper in the upper-left orange area, visible in Experiment I(d), Fig. 8.4. To correct these, a few labels are manually added around this zone (Experiment I(e), Fig. 8.4). As an effect, these misclassifications are removed: Experiment I(f), Fig. 8.4 shows that the upper-left orange area is now quite free of misclassifications (except one outlier point in the upper region). Most misclassifications occur now on the boundary of this area, apart from those occurring deep inside the blue area.

Final situation: The rightmost column in Figures 8.4 and 8.5 show, for all five experiments, the initial DBM and also the final DBM, after

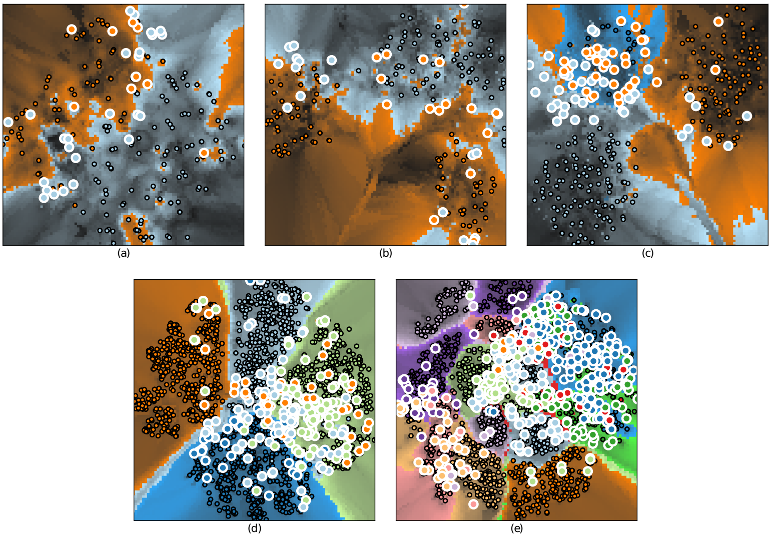


Figure 8.3: Initial decision boundary maps for the dataset/classifier pairs proposed for this experiment (a) Experiment I, (b) Experiment II, (c) Experiment III, (d) Experiment IV, (d) Experiment V.

manual labeling is finished. In general, the differences of the initial *vs* final maps are quite small, which is expected, as explained above, given the quite small number of manually added labels. However, the images for Experiment V (Fig. 8.5, right column) show an exception: The effects of manual label assignment are clearly visible, as the blue island located inside the green decision zone in the initial DBM shrinks considerably in the final DBM. This effect is desirable, as the images for Experiment V in the same figure show many misclassifications in this same region.

Effects of manual labeling: Figure 8.6 shows how training and validation accuracies for each experiment varied as batches of samples were labeled. For Experiments I, III, and V, the accuracy does, however, increase quite visibly – so, for these cases, we can say that manual labeling has a clear added value. For Experiment V, accuracy increases only slightly. For Experiment II, accuracy is relatively flat or can be seen as slightly decreasing. Overall, this indicates that having a consistent *gain* in accuracy as more samples are manually labeled is not an easy task in general.

8.4.1 *Comparison with automatic labeling*

Let us now study the accuracy gain delivered by manual labeling as compared to the baseline (no additional labels) and also as compared to

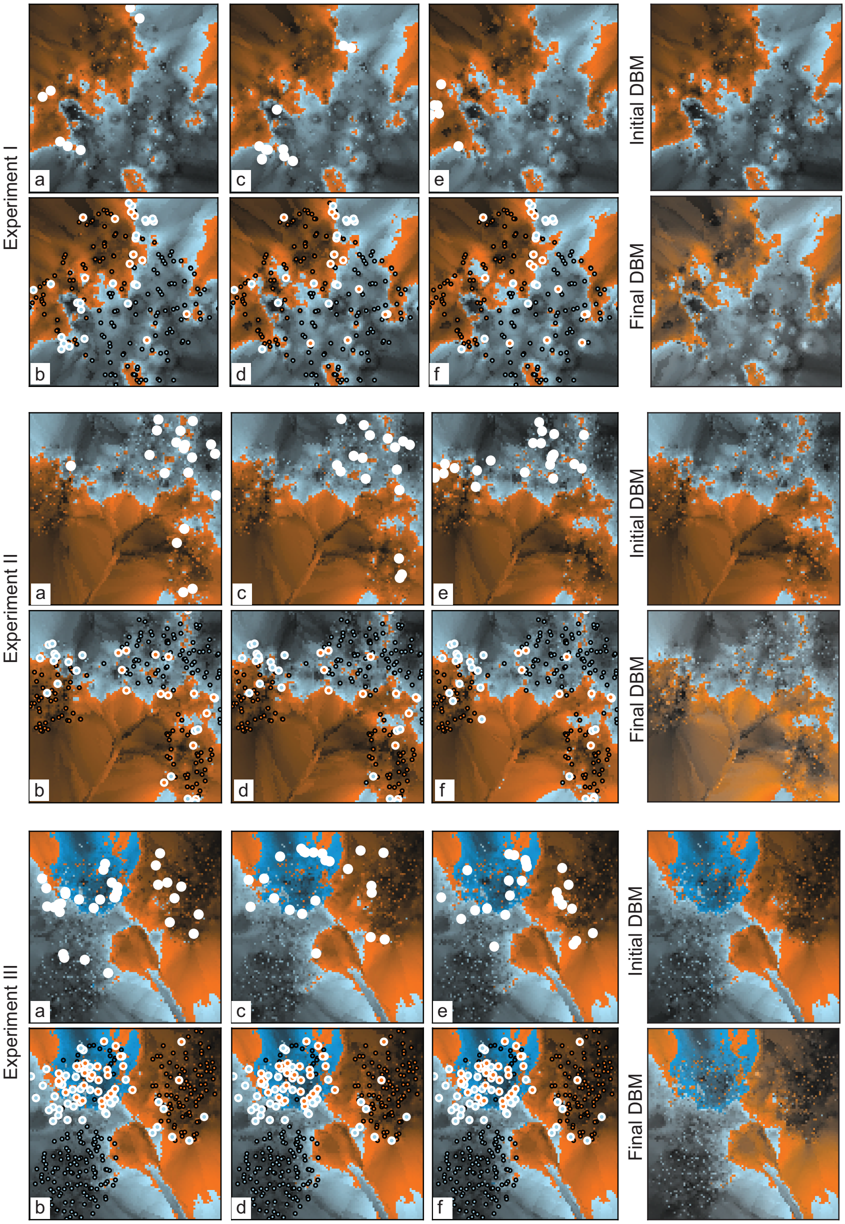


Figure 8.4: Sequences of manual label assignment, Experiments I-III. Images (a), (c) and (e) show the points selected to label as white disks atop of the DBMs. Images (b), (d) and (f) show the resulting DBMs after re-training the classifier with the new points added.

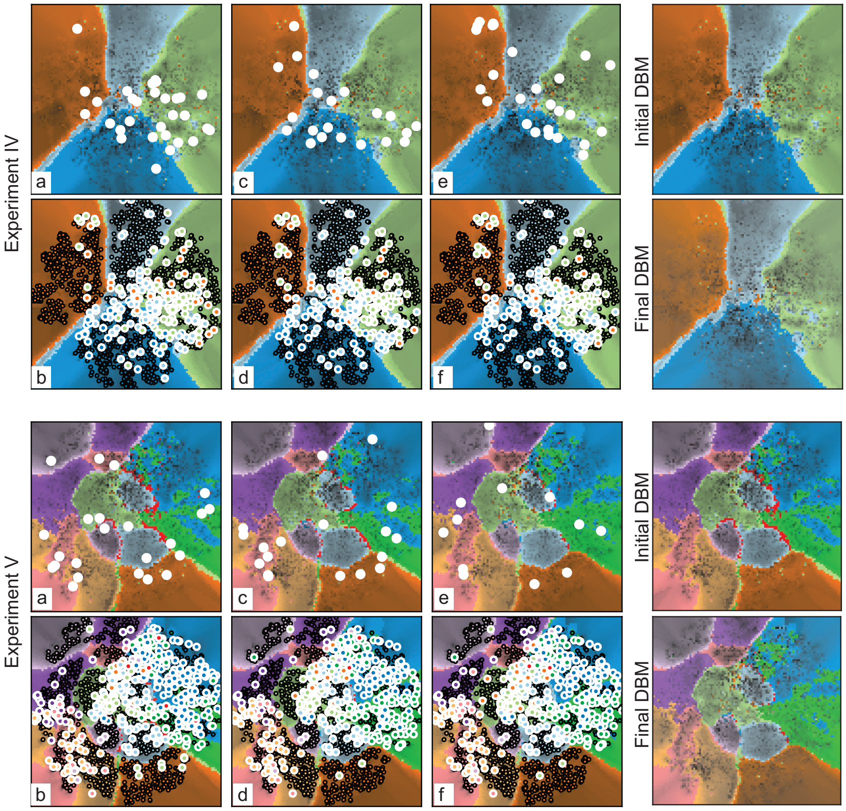


Figure 8.5: Sequences of manual label assignment, Experiments IV-V. Images (a), (c) and (e) show the points selected to label as white disks atop of the DBMs. Images (b), (d) and (f) show the resulting DBMs after retraining the classifier with the new points added.

automatic labeling (AL). Table 7 shows these figures. When compared to the baseline accuracy (Tab. 6), the resulting classifier accuracy after manual sample annotation shows an improvement of 4%, 16.2%, 22.5%, 1.5% and 0.2% for each of the five experiments, respectively (see Tab. 7, columns "Manual"). Experiments II and III, both using synthetic data and a MLP classifier, show the highest gains. However, as mentioned previously, this may be due to classifier overfitting in these cases. Nevertheless, the displayed gains justify that manual labeling supported by DBMs brings, in general, significant added value as opposed to using only the original labels.

Let us now compare manual labeling to fully automatic labeling (AL) under the two conditions mentioned in Sec. 8.3. In the first condition, we recall that AL is allowed to assign a label to every sample in U. The results of this process are shown in Tab. 7, columns "Automatic (full)".

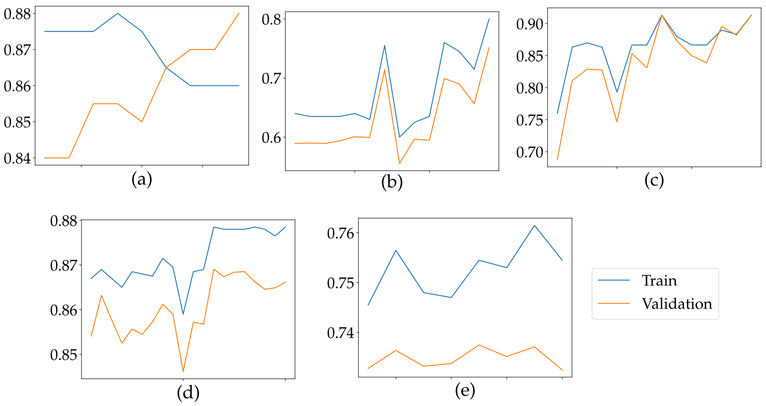


Figure 8.6: Graphs showing the classifier's accuracy change during labeling experiments. For each experiment, both training and validation accuracy are tracked. (a) Experiment I, (b) Experiment II, (c) Experiment III, (d) Experiment IV, (d) Experiment V.

We see that manual labeling performs quite similar to AL for Experiments I and III, but significantly better for Experiments II, IV, and V.

In the second condition, we allow AL to assign a label to every sample in U, but only k of them are randomly drawn to be used for the training of the classifier. We repeat this process of random drawing of labeled samples and classifier training-and-evaluation ten times. Table 7, columns "Automatic (k)" show the averaged results for each experiment. The number k of used labels equals the number of labels assigned manually, and is indicated in brackets in the first column. For Experiments I, II, and III, we see that AL slightly decreases in accuracy as compared to full labeling, while accuracy strongly increases *vs* full labeling for Experiments IV and V. Still, overall, our manual labeling performs better than AL under this condition.

For a final comparison, Tab. 7, columns "True labels (k)" show the results of training the classifier with true labels added to k points from U. As for the second condition, we did this experiment ten times and averaged results. This would be the optimal situation, equivalent to an AL method that perfectly guesses the labels for k such points. The obtained accuracies are now higher, but still, in general, slightly lower than those obtained using manual labeling.

Table 8 shows the number of correctly assigned labels for the manual labeling and AL under the two conditions. For the easiest case (Experiment I), we see that AL beats manual labeling. However, as Tab. 7 shows, the added value in terms of classifier accuracy is not too large, since this experiment uses a quite simple classifier (LR). In all other cases, we see that manual labeling guesses labels better than AL, even dramatically so for Experiments IV and V.

Table 7: Comparing achieved accuracies for each experiment with automatic label propagation.

Experiment (k)	Manual		Automatic (full)		Automatic (k)		True labels (k)	
	T	V	T	V	T	V	T	V
I (25)	86.0%	88.0%	88.5%	85.5%	87.4%	84.5%	87.3%	84.5%
II (58)	80.0%	75.1%	63.2%	60.7%	76.2%	70.6%	79.0%	72.5%
III (79)	91.3%	91.3%	91.6%	90.8%	71.6%	65.7%	81.7%	76.6%
IV (77)	87.8%	86.6%	23.5%	25.2%	60.1%	60.1%	87.1%	86.2%
V (53)	75.4%	73.2%	09.3%	10.6%	61.7%	62.0%	74.7%	72.9%

Table 8: Comparing the number of correct labels assigned manually and automatically.

Experiment (k)	Manual	Automatic (full)	Automatic (k)
I (25)	88.0%	94.0%	94.8%
II (58)	96.5%	92.5%	93.0%
III (79)	98.7%	94.3%	94.3%
IV (77)	89.6%	30.3%	28.3%
V (53)	84.9%	10.8%	12.0%

8.5 DISCUSSION

We discuss below several insights obtained during our experiments for assessing the working and added value of manual labeling assisted by DBMs.

Added value: As shown by the comparisons in Sec. 8.4.1, manual labeling aided by DBMs *does* bring visible added value in terms of both guessing the correct labels for data points and, more importantly we argue, obtaining classifiers with a higher accuracy. The difference (in terms of accuracy values) however does vary quite significantly as a function of the used classifier and classification problem (dataset). That is, for a simple problem, automatic methods perform quite well, as they arguably can easily "find their way" in the high-dimensional space. For more complex problems, however, the user's insights, obtained using DBMs, appear to beat the performance of automatic methods. We note that very similar trends have been exposed by the experiments of Benato *et al.* [13]. Since their experiments used different datasets, classifiers, and AL methods, we argue that our work strengthens the claim that visual analytic methods are a useful tool for classifier engineering in semi-supervised learning.

Local information: One important question is which are the visual hints that determine a user to assign a certain label to a certain point – or, alternatively, skip the point from labeling. As already explained, one such hint is the existence of clusters of misclassifications which are far away from decision boundaries, thus, which appear to be easily correctable by adding a few extra labeled points close to them. However, even when global data properties are preserved by projection methods, *i.e.* data points of the different classes are nicely split into different visual clusters, local properties can be more relevant to label assignment, such as local neighborhood information. For this reason, during our manual labeling experiments, the drawing of lines to indicate the true neighbors of an unlabeled point was very important when deciding how to label it.

User effort: Manual labeling is a time consuming task as users need to take a lot of information into consideration in order to decide which class to assign to a sample. For this reason, we limited our experiments to labeling only a few tens of points (see Tab. 7), and used the working set concept to offer only a few points at a time to the user to label, to limit visual clutter. While some of samples presented to the user were confusing to label, as explained earlier, most of the others did not seem to add much information to the classifier as they appeared to be deep into decision zones, as indicated by the distance-to-the-closest-decision boundary, visible as low luminance in the DBMs. One potential idea to reduce the labeling effort would be to offer selection mechanisms for users to remove from the labeling process samples which are deep into these zones, thereby concentrating the effort on arguably more important samples close to the decision boundaries. On the other hand, while we acknowledge that manual labeling is a difficult task, the proposed procedure allowed for a correct class attribution to samples, as Tab. 8 shows.

Another possible way to reduce user effort is to select the working set of samples offered to labeling in a more informed way than the order they come in the dataset (Sec. 8.2), using an active learning approach. This may focus the user's labeling effort in areas where it has the most impact. However, in this case, one may wonder what is the added value of seeing the DBM. Since we wanted (as also explained next under "Ground truth hints") to test the added-value of DBMs independently on other mechanisms, we did not investigate this path. Nevertheless, seeing whether a combination of active learning mechanisms and the DBM can provide added value is an interesting future work direction.

Ground truth hints: During our experiments, the user was not allowed to look at the actual features (although we did implement the facility to display this information in our tool). We did this so as to better assess how the DBMs *by themselves* can assist manual labeling and classifier engineering. Note that this is in contrast to Benato *et al.*, where users actively used tooltips to inspect the unlabeled images during manual label propagation. Still, even under this restrictive condition, our results are arguably quite good, thereby justifying the added value of DBMs *in isolation*. For image data, it is reasonable to expect that adding the option to see the images during label propagation would only increase the performance of the human-in-the-loop method. Doing this experiment is left to future work.

Extensions: Additional experiments with a human-in-the-loop may benefit from VA tools, for example in the case of imbalanced datasets. In such a scenario, we conjecture that human intuition could help correctly labeling data better than automatic methods, which would be biased by the class having most samples. As for the previous point, we leave this investigation to future research.

9

CONCLUSION

Throughout this manuscript, we investigated and developed methods for the visual exploration of the decision zones and decision boundaries of Machine Learning classifiers, with the aim of helping the classifier engineer to better understand how such classifiers partition their input data space into decision zones and, when possible, influence this partition to improve the performance of the respective classifier.

The main contribution of this thesis is the proposal of a visualization method that constructs explicit and dense visual representations of decision zones and their respective boundaries. We augmented this visualization by several mechanisms in order to provide additional information on the explored data space, and we used the resulting techniques to construct a visual analytics (VA) tool and workflow that supports the process of label creation in semi-supervised machine learning scenarios. The proposed techniques, together with their deployment in the VA tool, are our answer to the research questions raised in Section 1.4.

We next briefly discuss each chapter of this thesis next, summarizing their relevance to our research objectives. We end this chapter with some pointers to future work and possible improvements to our work.

9.1 DEEP FEATURE EXTRACTION EVALUATION

In Chapter 3, transfer learning was used to solve a practical problem of planktonic image classification. Besides achieving a high accuracy on the task at hand, we wanted to investigate the impact of dataset similarity, *i.e.* different sources S_i, and of classifier architecture, *i.e.* different C_i, on classifying our local data. This analysis was conducted by computing accuracy metrics for different pairs of classifier-dataset combinations and by using standard, but very simple, visualization tools such as tables, confusion matrices, and aggregated bar charts.

This chapter showed that, using the currently available simple visualization tools, one can still construct and validate a classifier than produces good results for a given practical problem. In the same time, this work highlighted several challenges and limitations that such visualization tools have in terms of limited insights that they produce and questions that they cannot answer. As such, the material here provides us with a practical justification for the need for developing more powerful visualization tools for classifier engineering.

9.2 DECISION BOUNDARY MAPS

While confusion matrices, used in Chapter 3, can relate different classes in a single image, indicating which ones are harder for a classifier to discriminate, the information provided is far too simple and cannot relate individual samples, which have to be manually explored.

In Chapter 4 we proposed an image-based method to depict how the decision boundaries induced by a classifier are related. In contrast to the aggregated visualization methods discussed in Chapter 3, our proposal is an observation-centric method which dedicates space in the visualization to every sample. This way, observation-related patterns, such as clusters of similar samples and/or outliers, can be readily explored. Our proposed visualization literally “fills” the empty gaps that exist in scatterplots created by dimensionality reduction methods by telling the user how a classifier would behave for data that would map to such gaps. To our knowledge, our decision boundary maps, or DBMs (as we call our method) is the first method to explicitly compute and visualize the shape and location of decision zones and their boundaries for any classifier applied on any high-dimensional dataset.

As shown in subsequent chapters through a number of examples, our method is indeed capable of providing relevant insights into the classifier-dataset relationship, allowing for both a global view of the decision zones and providing local details.

9.3 IMPACT OF DIRECT PROJECTIONS ON DBMS

The construction of decision boundary maps depends on two key techniques: a direct projection and an inverse projection. In Chapter 5, we evaluate how DBMs depend on the choice of the direct projection technique being used. In total, we studied 28 projection techniques for this task, allowing for a broad comparison. Different classifier conditions were also explored, starting from a linear classifier that is capable of classifying a simple dataset with 100% accuracy up to a Convolutional Neural Network applied to a more complex task of classifying natural image data.

The results of this study pointed out to a small set of methods, namely t-SNE and UMAP, which appear to be the best for the construction of DBMs for all studied classifiers and datasets. In the same time, we highlighted several problems that appear during DBM construction – such as jagged boundaries and spurious islands – and which cannot be fully removed by the change of the projection technique used. We address these problems further in Chapter 7.

9.4 IMPACT OF INVERSE PROJECTION ON DBM FORMATION

In Chapter 6, we study the second key ingredient of computing DBMs, namely the inverse projection technique being used. For this, we considered first two inverse projections available in the literature, namely iLAMP and RBF. The results obtained with these methods indicate that DBMs still suffer from problems such as the aforementioned spurious islands. Equally importantly, the execution time of both these inverse projections is very large, which prohibits the use of the so-created DBMs in interactive visual analytics contexts.

We address the above limitations by proposing a novel method to compute inverse projections based on deep learning the inverse transformation implied by a direct projection method. We compare the three alternatives (iLAMP, RBF, and our deep learned inverse projection called NNinv), and show that our proposed method produces better DBMs, and with a significantly lower computational cost, than iLAMP and RBF. Besides DBM computation, our inverse projection can be generically used in any other application which requires inverse projection capabilities.

9.5 VISUAL REFINEMENTS OF DBMS

Following the work of the previous two chapters, we converge with the design of DBMs based on t-SNE or UMAP as direct projections, and NNinv as inverse projection respectively. However, even when using these elements, DBMs still suffer from imprecisions caused by the inherent limitations of both direct and inverse projections.

In Chapter 7, we refine both the construction and visualization of DBMs as follows. First, we propose a filtering strategy that removes points from the projection which cause the spurious island problems. While slightly reducing the amount of data shown to the user, the remaining data is shown much more accurately, therefore obtaining an overall better, less confusing, visualization. Secondly, we enhance DBMs to show, for each pixel, the distance to the closest decision boundary. This way, users can readily separate the areas which are deep inside decision zones (therefore, less interesting for classifier engineering as these are arguably easy to decide upon) from the areas closer to decision boundaries (where arguably a classifier has more problems, thus, where the user's effort for improving the classifier should concentrate). We provide three different heuristics for computing the distance-to-boundary, which offer different trade-offs between computational speed and accuracy.

With the techniques presented in this chapter, we conclude our answers to the first research question posed in Section 1.4, that we repeat below:

How to use visual analytics to get more insights into a classifier's performance?

9.6 END TO END APPLICATION

In Chapter 8, we focus on our second research question presented in Section 1.4, which we repeat below:

How to use visual analytics to improve a classifer?

We address this question by presenting a visual analytics tool, and corresponding workflow, that combines classifier training, testing, and validation, with a semi-supervised learning process based on manual sample annotation (labeling). Central to this process is the usage of DBMs and various other visualization mechanisms (tooltips, nearest neighbors) to help the user in the selection and labeling of samples.

We demonstrate our approach by using our proposed visual analytics tool and workflow in the process of semi-supervised learning for five scenarios that combine five different datasets with three classifier techniques. In each experiment, we allow the user to label only a few tens of samples, and next test the increase of accuracy due to these samples as compared to the original training set (baseline), but also compared to using a fully automatic label propagation (AL) procedure.

The results of these experiments show several insights. On the positive side, our manual labeling assisted by DBMs produces significantly more correctly labeled samples than AL, which, in turn, lead to classifiers having a better performance than the AL-trained ones. On the less positive side, the overall increases in performance as compared to the baseline training are limited. Our results strengthen independently executed complementary recent research in using visual analytics to assist semi-supervised learning, and, as such, motivate the pursuing of future work in this area.

9.7 FUTURE WORK

Several directions for future work are possible, as follows.

On the practical side, one interesting direction consists in applying our techniques for visualizing decision boundaries in different machine learning settings, such as on regression problems. This would extend the application area, and therefore the practical added value, of DBMs to different problems.

On the technical side, it is definitely of interest to revisit the basic DBM computation algorithm proposed in Chapter 4. A major point of interest would be to re-think the algorithm so as to minimize *errors in* *the entire DBM.* Right now, our algorithm still suffers from errors of both the direct and the inverse projection, and these projections (and their errors) are computed based on a typically small number of sample points. Since DBMs aim to visualize the entire high-dimensional space (in the limit), they use a very large number of points in their construction. Hence, it should be possible to cast the DBM construction problem as an explicit model that joins the costs of direct and inverse projection for all the points which ultimately create the DBM. We believe that this would lead to DBMs of significantly higher quality.

Finally, half-way between the technical and practical sides, it is interesting to further refine the VA workflow and associated tools based on DBMs aimed at supporting classifier engineering. Our current approach was minimal, in the sense of using only a single-resolution view of DBMs, with only a few additional interactive investigation mechanisms such as tooltips and nearest neighbor markers. It is well known that VA tools excel in their task when offering a wide range of finely-tuned exploration options to the user. This would translate in our context to providing ways to compute DBMs in a multiresolution fashion, *e.g.* to allow users to seamlessly zoom-in-and-out in specific areas of the data space, or compute additional metrics that rank samples for manual labeling and inform the user on this ranking. In the long run, providing real-time computation and visualization of DBMs would allow a truly immersive experience, where one could literally label and/or move sample points in the projection and see how the decision zones and their boundaries change, thereby getting an intimate feeling of how a classifier works to construct these important, but still elusive, boundaries.

BIBLIOGRAPHY

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[2] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H. T. Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA:, 2012.

[3] A. Adadi and M. Berrada. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access*, 6, 2018. DOI:10.1109/ACCESS.2018.2870052.

[4] H. A. Al-Barazanchi, A. Verma, and S. Wang. Performance evaluation of hybrid cnn for sipper plankton image calssification. In *2015 Third International Conference on Image Information Processing (ICIIP)*, pages 551–556. IEEE, 2015.

[5] 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.

[6] W. P. Amorim, A. X. Falcão, J. a. P. Papa, and M. H. Carvalho. Improving semi-supervised learning through optimum connectivity. *Pattern Recogn.*, 60(C):72–85, December 2016. ISSN 0031-3203.

[7] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *J. of the ACM*, 45(6):891–923, 1998.

[8] M. Aupetit. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing*, 10(7-9): 1304–1330, 2007.

[9] F. Aurenhammer. Voronoi diagrams: A survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23:345–405, 1991.

[10] C. Azodi, J. Tang, and S. Shiu. Opening the black box: Interpretable machine learning for geneticists. *Trends in Genetics*, 6 (26):442–455, 2020.

[11] 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.

[12] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *J. Mach. Learn. Res.*, 7:2399–2434, December 2006. ISSN 1532-4435.

[13] B. C. Benato, A. C. Telea, and A. X. Falcão. Semi-supervised learning with interactive label propagation guided by feature space projections. In *2018 31st SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, pages 392–399. IEEE, 2018.

[14] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 17–36, 2012.

[15] M. de Berg, M. van Kreveld, M. Overmars, and O. Cheong-Schwarzkopf. *Computational Geometry – Algorithms and Applications.* Springer, 2000.

[16] C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[17] M. B. Blaschko, G. Holness, M. A. Mattar, D. Lisin, P. E. Utgoff, A. R. Hanson, H. Schultz, E. M. Riseman, M. E. Sieracki, W. M. Balch, et al. Automatic in situ identification of plankton. In *2005 Seventh IEEE Workshops on Applications of Computer Vision (WACV/MOTION'05)-Volume 1*, volume 1, pages 79–86. IEEE, 2005.

[18] L. Bottou. Stochastic gradient tricks. In G. Montavon, G. B. Orr, and K. R. Müller, editors, *Neural Networks, Tricks of the Trade, Reloaded*, Lecture Notes in Computer Science (LNCS 7700), pages 430–445. Springer, 2012.

[19] B. Broeksema, A. Telea, and T. Baudel. Visual analysis of multi?dimensional categorical data sets. *Computer Graphics Forum*, 32(8):158–169, 2013.

[20] 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.

[21] D. B. Carr, R. J. Littlefield, W. L. Nicholson, and J. S. Littlefield. Scatterplot matrix techniques for large *n*. *Journal of the American Statistical Association*, 82(398):424–436, 1987.

[22] 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.

[23] F. Chollet. Keras machine learning framework, 2018. `https://github.com/fchollet/keras`.

[24] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The Loss Surfaces of Multilayer Networks. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 192–204. PMLR, 2015.

[25] L. Costa and R. Cesar. *Shape analysis and classification*. CRC Press, 2000.

[26] Cowen, R. K., S. Sponaugle, K. Robinson, and J. Luo. PlanktonSet 1.0: Plankton imagery data collected from F.G. Walton Smith in Straits of Florida from 2014-06-03 to 2014-06-06 and used in the 2015 National Data Science Bowl. 2015. DOI 10.7289/V5D21VJD.

[27] J. Cunningham and Z. Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *JMLR*, 16:2859–2900, 2015.

[28] J. Dai, R. Wang, H. Zheng, G. Ji, and X. Qiao. Zooplanktonet: deep convolutional network for zooplankton classification. In *OCEANS 2016-Shanghai*, pages 1–6. IEEE, 2016.

[29] T. N. Dang and L. Wilkinson. Scagexplorer: Exploring scatterplots by their scagnostics. In *Visualization Symposium (PacificVis), 2014 IEEE Pacific*, pages 73–80. IEEE, 2014.

[30] S. Dasgupta. Experiments with random projection. In *Proc. of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 143–151. Morgan Kaufmann, 2000.

[31] O. Delalleau, Y. Bengio, and N. Le Roux. Efficient non-parametric function induction in semi-supervised learning. In *AISTATS*, volume 27 number 28, page 100. Citeseer, 2005.

[32] D. Dheeru and E. Karra Taniskidou. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[33] S. Dieleman, J. De Fauw, and K. Kavukcuoglu. Exploiting cyclic symmetry in convolutional neural networks. *arXiv preprint arXiv:1602.02660*, 2016.

[34] 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.

[35] F. K. Dosilovi'c, M. Brci'c, and N. Hlupi'c. Explainable artificial intelligence: A survey. In *Proc. 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 210–215, 2018.

[36] D. Engel, L. Hüttenberger, 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–Leibniz-Zentrum fuer Informatik, 2012.

[37] M. Espadoto, R. Martins, A. Kerren, N. Hirata, and A. Telea. Towards a quantitative survey of dimension reduction techniques. *IEEE TVCG*, 2019. doi:10.1109/TVCG.2019.2944182.

[38] M. Espadoto, F. C. M. Rodrigues, and A. Telea. Visual analytics of multidimensional projections for constructing classifier decision boundary maps. In *Proc. IVAPP*. SciTePress, 2019.

[39] M. Espadoto, A. Falcao, N. Hirata, and A. Telea. Improving neural network-based multidimensional projections. In *Proc. IVAPP*. SciTePress, 2020.

[40] M. Espadoto, N. Hirata, and A. Telea. Deep learning multidimensional projections. *J. Information Visualization*, 2020. `https://doi.org/10.1177/1473871620909485`.

[41] M. Espadoto, N. Hirata, and A. Telea. Deep learning multidimensional projections. *Information Visualization*, 2020. DOI:10.1177/1473871620909485.

[42] M. Espadoto, F. C. M. Rodrigues, N. S. T. Hirata, R. Hirata Jr., and A. C. Telea. Deep Learning Inverse Multidimensional Projections. In *Proc. EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association, 2019.

[43] 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.

[44] C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *ACM SIGMOD Newsletter*, 24(2):163–174, 1995.

[45] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.

[46] 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.

[47] J. M. Flores, M. Fischer, A. Telea, and L. Linsen. Scatterplot summarization by constructing fast and robust principal graphs from skeletons. In *Proc. IEEE PacificVis*, 2019.

[48] I. K. Fodor. A survey of dimension reduction techniques. Technical report, US Dept. of Energy, Lawrence Livermore National Labs, 2002. Tech. report UCRL-ID-148494.

[49] R. Garcia, A. Telea, I. da Silva, J. Torresen, and J. Comba. A task-and-technique centered survey on visual analytics for deep learning model engineering. *Computers and Graphics*, 77:30–49, 2018.

[50] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. IEEE CVPR*, pages 580–587, 2014.

[51] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[52] M. Guillaumin, T. Mensink, J. Verbeek, and C. Schmid. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *2009 IEEE 12th International Conference on Computer Vision*, pages 309–316, Sept 2009.

[53] L. Hamel. Visualization of support vector machines with unsupervised learning. In *Proc. Computational Intelligence and Bioinformatics and Computational Biology (CIBCB)*. IEEE, 2006.

[54] C. Hansen and C. Johnson. *The visualization handbook*. Elsevier, 2005.

[55] R. M. Haralick, K. Shanmugam, et al. Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, (6):610–621, 1973.

[56] T. Hastie and W. Stuetzle. Principal curves. *J. American Statistical Association*, 84(406):502–516, 1989.

[57] 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.

[58] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[59] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[60] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proc. ACM SIGGRAPH*, pages 277–286, 1999.

[61] 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.

[62] X. Hu, H. Cammann, H. A. Meyer, K. Miller, K. Jung, and C. Stephan. Artificial neural networks and prostate cancer—tools for diagnosis and management. *Nature Reviews Urology*, 10(3): 174–182, 2013.

[63] A. Hyvarinen. Fast ICA for noisy data using gaussian moments. In *Proc. IEEE ISCAS*, volume 5, pages 57–61, 1999.

[64] A. Inselberg and B. Dimsdale. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In *Proc. IEEE VIS*, pages 361–378, 1990.

[65] J. C. S. Jacques, C. R. Jung, and S. R. Musse. A background subtraction model adapted to illumination changes. In *2006 International Conference on Image Processing*, pages 1817–1820. IEEE, 2006.

[66] P. Joia, D. Coimbra, J. A. Cuminato, F. V. Paulovich, and L. G. Nonato. Local affine multidimensional projection. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2563–2571, 2011.

[67] I. T. Jolliffe. Principal Component Analysis and Factor Analysis. In *Principal component analysis*, pages 115–128. Springer, 1986.

[68] J. Kehrer and H. Hauser. Visualization and visual analysis of multifaceted scientific data: A survey. *IEEE TVCG*, 19(3):495–513, 2013.

[69] D. Keim, G. Andrienko, J. D. Fekete, C. Görg, J. Kohlhammer, and G. Melan con. Visual analytics: Definition, process, and challenges. In *Information Visualization*, pages 154–175. Springer, 2008.

[70] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980v9 [cs.LG]*, 2014.

[71] D. P. Kingma and M. Welling. Auto-encoding variational Bayes, 2013. arXiv:1312.6114 [cs.ML].

[72] I. Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1):89–109, 2001.

[73] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[74] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[75] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database, 2010. AT&T Labs [Online]. Available: `http://yann.lecun.com/exdb/mnist`.

[76] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521 (7553):436–444, 2015.

[77] D. J. Lehmann, G. Albuquerque, M. Eisemann, M. Magnor, and H. Theisel. Selecting coherent and relevant plots in large scatterplot matrices. *Computer Graphics Forum*, 31(6):1895–1908, 2012.

[78] R. van Liere and W. De Leeuw. GraphSplatting: Visualizing graphs as continuous fields. *Visualization and Computer Graphics, IEEE Transactions on*, 9:206– 212, 05 2003.

[79] J. Liu, M. Li, Q. Liu, H. Lu, and S. Ma. Image annotation via graph learning. *Pattern Recognition*, 42(2):218 – 228, 2009. ISSN 0031-3203. Learning Semantics from Multimedia Content.

[80] 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.

[81] L. van der Maaten. Learning a parametric embedding by preserving local structure. In *Proc. 12th Intl. Conf. on Artificial Intelligence and Statistics*, 2009.

[82] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *JMLR*, 9(Nov):2579–2605, 2008.

[83] L. van der Maaten and E. Postma. Dimensionality reduction: A comparative review, 2009. Tech. report TiCC TR 2009-005, Tilburg University, Netherlands.

[84] C. D. Manning, H. Schütze, and P. Raghavan. *Introduction to Information Retrieval*, volume 39. Cambridge University Press, 2008.

[85] R. Martins, D. Coimbra, R. Minghim, and A. C. Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014.

[86] R. Martins, R. Minghim, and A. C. Telea. Explaining neighborhood preservation for multidimensional projections. In *Proc. CGVC*, pages 121–128. Eurographics, 2015.

[87] L. McInnes and J. Healy. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv:1802.03426v1 [stat.ML]*, 2018.

[88] M. A. Migut, M. Worring, and C. J. Veenman. Visualizing multidimensional decision boundaries in 2D. *Data Mining and Knowledge Discovery*, 29(1):273–295, 2015.

[89] 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.

[90] T. Munzner. *Visualization analysis and design.* CRC Press, 2015.

[91] 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.

[92] T. Ojala, M. Pietikäinen, and T. Mäenpää. Gray scale and rotation invariant texture classification with local binary patterns. In *European Conference on Computer Vision*, pages 404–420. Springer, 2000.

[93] E. C. Orenstein and O. Beijbom. Transfer learning and deep feature extraction for planktonic image data sets. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1082–1088. IEEE, 2017.

[94] E. Osuna, R. Freund, and F. Girosit. Training support vector machines: an application to face detection. In *Computer vision and pattern recognition, 1997. Proceedings., 1997 IEEE computer society conference on*, pages 130–136. IEEE, 1997.

[95] N. Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9 (1):62–66, 1979.

[96] C. W. A. M. van Overveld and J. J. van Wijk. Preset based interaction with high dimensional parameter spaces. In *Data Visualization*, pages 391–406. Springer, 2003.

[97] U. Ozertem and D. Erdogmus. Locally defined principal curves and surfaces. *JMLR*, 12:1249–1286, 2011.

[98] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[99] A. V. Pandey, J. Krause, C. Felix, J. Boy, and E. Bertini. Towards understanding human similarity perception in the analysis of large sets of scatter plots. In *Proc. ACM CHI*, pages 3659–3669, 2016.

[100] 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.

[101] F. V. Paulovich, C. T. Silva, and L. G. Nonato. Two-phase mapping for projecting massive data sets. *IEEE TVCG*, 16(6):1281–1290, 2010.

[102] F. V. Paulovich, D. M. Eler, J. Poco, , C. P. Botha, R. Minghim, and L. G. Nonato. Piecewise laplacian-based projection for interactive data exploration and organization. *Computer Graphics Forum*, 30 (3):1091–1100, 2011.

[103] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[104] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[105] E. Pekalska, D. de 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.

[106] N. Pezzotti, T. Höllt, B. Lelieveldt, E. Eisemann, and A. Vilanova. Hierarchical stochastic neighbor embedding. *Computer Graphics Forum*, 35(3):570–580, 2016.

[107] N. Pezzotti, B. P. Lelieveldt, L. van der Maaten, T. Höllt, E. Eisemann, and A. Vilanova. Approximated and user steerable t-SNE for progressive visual analytics. *IEEE TVCG*, 23(7):1739–1752, 2017.

[108] N. Pise and P. Kulkarni. A survey of semi-supervised learning methods. In *Proc. Intl. Conf. on Comp. Intell. and Security*, 2008.

[109] K. Polat and S. Güneş. Breast cancer diagnosis using least square support vector machine. *Digital Signal Processing*, 17(4):694–701, 2007.

[110] G. Pölzlbauer. Survey and comparison of quality measures for self-organizing maps. In *Proc. Workshop on Data Analysis (WDA)*, pages 67–82, 2004.

[111] V. Prakash and L. Nithya. A survey on semi-supervised learning techniques. *International Journal of Computer Trends and Technology*, 8(1):25–29, 2014.

[112] O. Py, H. Hong, and S. Zhongzhi. Plankton classification with deep convolutional neural networks. In *2016 IEEE Information Technology, Networking, Electronic and Automation Control Conference*, pages 132–136. IEEE, 2016.

[113] 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.

[114] P. Rauber, R. da Silva, S. Feringa, M. Celebi, A. Falcao, and A. Telea. Interactive image feature selection aided by dimensionality reduction. In *Proc. EuroVA*. Eurographics, 2015.

[115] 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.

[116] 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.

[117] R. A. Rensink and G. Baldridge. The perception of correlation in scatterplots. *Computer Graphics Forum*, 29(3):1203–1210, 2010.

[118] M. T. Ribeiro, S. Singh, and C. Guestrin. Why should i trust you? explaining the predictions of any classifier. In *Proc. ACM KDD*, 2016.

[119] F. C. M. Rodrigues, M. Espadoto, R. H. Jr, and A. Telea. Constructing and visualizing high-quality classifier decision boundary maps. *Information*, 10(9):280–297, 2019.

[120] 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.

[121] F. C. M. Rodrigues., N. S. T. Hirata., A. A. Abello., L. T. de la Cruz., R. M. Lopes., and R. H. Jr.. Evaluation of transfer learning scenarios in plankton image classification. In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP,*, pages 359–366. SciTePress, 2018.

[122] S. T. Roweis and L. L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[123] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. DOI 10.1007/s11263-015-0816-y.

[124] E. P. dos Santos Amorim, E. V. Brazil, J. Daniels, P. Joia, L. G. Nonato, and M. C. Sousa. ilamp: Exploring high-dimensional spacing through backward multidimensional projection. In *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 53–62. IEEE, 2012.

[125] B. Schölkopf, A. Smola, and K. Müller. Kernel principal component analysis. In *Proc. International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[126] T. Schreck, T. von Landesberger, and S. Bremm. Techniques for precision-based visual analysis of projected data. *Information Visualization*, 9(3):181–193, 2010.

[127] A. Schulz, A. Gisbrecht, and B. Hammer. Using discriminative dimensionality reduction to visualize classifiers. *Neural Processing Letters*, 42(1):27–54, 2015.

[128] F. Sebastiani. Machine learning in automated text categorization. *ACM computing surveys (CSUR)*, 34(1):1–47, 2002.

[129] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms.* Cambridge university press, 2014.

[130] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv:1703.00810v3 [cs.LG]*, 2017.

[131] R. da Silva, P. Rauber, R. Martins, R. Minghim, and A. Telea. Attribute-based visual explanation of multidimensional projections. In *Proc. EuroVA*, pages 95–102. Eurographics, 2015.

[132] V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Proc. NIPS*, volume 15, pages 721–728, 2003.

[133] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[134] B. W. Silverman. *Density Estimation for statistics and data analysis.* Monographs on Statistics and Applied Probability. Champan and Hall, 1986.

[135] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[136] V. Sindhwani, P. Niyogi, and M. Belkin. Beyond the point cloud: From transductive to semi-supervised learning. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 824–831, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5.

[137] M. Sips, B. Neubert, J. P. Lewis, and P. Hanrahan. Selecting good views of high-dimensional data using class consistency. *Computer Graphics Forum*, 28(3):831–838, 2009.

[138] D. Smilkov and S. Carter. TensorFlow playground, 2018. `https://playground.tensorflow.org`.

[139] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing and Management*, 45:427–437, 2009.

[140] C. Sorzano, J. Vargas, and A. Pascual-Montano. A survey of dimensionality reduction techniques, 2014. arXiv:1403.2877 [stat.ML].

[141] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15 (56):1929–1958, 2014.

[142] R. Strzodka and A. Telea. Generalized distance transforms and skeletons in graphics hardware. In *Proc. VisSym*. IEEE, 2004.

[143] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[144] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[145] A. Telea. Image-based visualization of Voronoi diagrams, 2014. `https://www.cs.rug.nl/svcg/DataVisualizationBook/Sp3`.

[146] A. Telea and O. Ersoy. Image-based edge bundles: Simplified visualization of large graphs. *CGF*, 29(3):543–551, 2010.

[147] A. Telea and L. Voinea. An open framework for CVS repository querying, analysis and visualization. In *Proc. Mining Software Repositories*. ACM, 2006.

[148] A. Telea and J. van Wijk. Visualization of generalized Voronoi diagrams. In *Proc. VisSym*. Springer, 2001.

[149] A. C. Telea. Combining extended table lens and treemap techniques for visualizing tabular data. In *Proc. EuroVis*, pages 120–127, 2006.

[150] A. C. Telea. *Data visualization – Principles and Practice*. CRC Press, 2 edition, 2015.

[151] J. B. Tenenbaum, V. D. Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[152] J. Thomas and K. A. Cook. *Illuminating the Path: The Research and Development Agenda for Visual Analytics*. National Visualization and Analytics Ctr, 2005.

[153] K. M. Ting. *Encyclopedia of machine learning*. Springer, 2011.

[154] J. W. Tukey and P. A. Tukey. Computer graphics and exploratory data analysis: An introduction. *The Collected Works of John W. Tukey: Graphics: 1965-1985*, 5:419, 1988.

[155] J. J. van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proc. IEEE InfoVis*, pages 73–78, Los Alamitos, CA, 1999. IEEE Press.

[156] J. L. Vermeulen, A. Hillebrand, and R. Geraerts. A comparative study of k?nearest neighbour techniques in crowd simulation. *Computer Animation & Virtual Worlds*, 28(3-4), 2017.

[157] E. Vernier, R. Garcia, I. da Silva, J. Comba, and A. Telea. Quantitative evaluation of time-dependent multidimensional projection techniques. *Computer Graphics Forum*, 39(20), 2020.

[158] F. Wang and C. Zhang. Label propagation through linear neighborhoods. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):55–67, Jan 2008. DOI 10.1109/TKDE.2007.190672.

[159] J. J. van Wijk and A. Telea. Enridged contour maps. In *Proceedings Visualization, 2001. VIS'01.*, pages 69–543. IEEE, 2001.

[160] L. Wilkinson, A. Anand, and R. Grossman. High-dimensional visual analytics: Interactive exploration guided by pairwise views of point distributions. *IEEE TVCG*, 12(6):1363–1372, 2006.

[161] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms, 2017. arXiv:1708.07747v2 [cs.LG].

[162] H. Xie, J. Li, and H. Xue. A survey of dimensionality reduction techniques based on random projection, 2017. arXiv:1706.04371 [cs.LG].

[163] 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.

[164] H. Yin. Nonlinear dimensionality reduction and data visualization: A review. *Intl. Journal of Automation and Computing*, 4(3): 294–303, 2007.

[165] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[166] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[167] 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.

[168] 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.

[169] X. Zhu and A. Goldberg. *Introduction to Semi-Supervised Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan Claypool, 2009.

[170] E. Zihni, V. I. Madai, M. Livne, I. Galinovic, A. A. Khalil, J. B. Fiebach, and D. Frey. Opening the black box of artificial intelligence for clinical decision support: A study predicting stroke outcome. *PloS ONE*, 15(4), 2020.

[171] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2): 265–286, 2006.

[172] M. van der Zwan, V. Codreanu, and A. Telea. CUBu: Universal real-time bundling for large graphs. *IEEE TVCG*, 22(12):2550–2563, 2016.

CURRICULUM VITAE

Francisco Caio Maia Rodrigues was born on the 4th of May of 1989 in Fortaleza, Brazil. In 2011 he received his bachelor degree in Computer Science from Federal University of Ceará. He received his master degree in Computer Science from the same university, under the supervision of Professor J. B. Cavalcante Neto. From 2016 to 2020 he was PhD student in a double degree programme between University of São Paulo and University of Groningen, under the supervision of Professor R. Hirata Jr. and Professor A. C. Telea, which resulted in this thesis.

ACKNOWLEDGMENTS

I would first like to thank my supervisors, Professor Alexandru Telea and Professor Roberto Hirata Jr., for providing guidance and feedback during the whole PhD proccess.

I would like to acknowledge the members of the assessement committe for reviewing this manuscript, and I sincerely hope that they could found it useful for their own research interests, or at least enjoyed reading about a different subject.

I would also like to thank my friends and colleagues, both in Brazil and in the Netherlands, for the stimulating conversations, amazing experiences we shared and for enduring my companionship.

In addition, I would like to thank my family for all the support.

I would like to thank CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil – Finance Code 001) for the financial 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 August 24, 2020 (`classicthesis`).