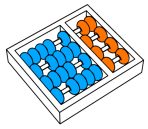
Universidade Estadual de Campinas
Instituto de Computação



Paulo Eduardo Rauber

Visual Analytics Applied to Image Analysis

Análise Visual Aplicada à Análise de Imagens

CAMPINAS
2017

Paulo Eduardo Rauber

Visual Analytics Applied to Image Analysis

Análise Visual Aplicada à Análise de Imagens

Tese apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação no âmbito do acordo de Cotutela firmado entre a Unicamp e a Universidade de Groningen.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Computer Science under the double-diploma agreement between Unicamp and University of Groningen.

Supervisores/Orientadores: Prof. Dr. Alexandre Xavier Falcão e Prof. Dr. Alexandru Cristian Telea
Co-supervisores/Coorientadores: Prof. Dr. Pedro Jussieu de Rezende e Prof. Dr. Johannes Bernardus Theodorus Maria Roerdink

Este exemplar corresponde à versão final da Tese defendida por Paulo Eduardo Rauber e orientada pelo Prof. Dr. Alexandre Xavier Falcão e Prof. Dr. Alexandru Cristian Telea.

CAMPINAS
2017

Agência(s) de fomento e nº(s) de processo(s): FAPESP, 2012/24121-9; CAPES; Ubbo Emmius Fund

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

R19v

Rauber, Paulo Eduardo, 1989-
Visual analytics applied to image analysis / Paulo Eduardo Rauber. – Campinas, SP : [s.n.], 2017.

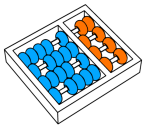
Orientadores: Alexandre Xavier Falcão e Alexandru Cristian Telea.
Coorientadores: Pedro Jussieu de Rezende e Johannes Bernardus Theodorus Maria Roerdink.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.
Em cotutela com: Universidade de Groningen.

1. Análise de imagem. 2. Visualização de informação. 3. Aprendizado de máquina. I. Falcão, Alexandre Xavier,1966-. II. Telea, Alexandru Cristian. III. Rezende, Pedro Jussieu de,1955-. IV. Roerdink, Johannes Bernardus Theodorus Maria,1954-. V. Universidade Estadual de Campinas. Instituto de Computação. VII. Título.

Informações para Biblioteca Digital

Título em outro idioma: Análise visual aplicada à análise de imagens
Palavras-chave em inglês:
Image analysis
Information visualization
Machine learning
Área de concentração: Ciência da Computação
Titulação: Doutor em Ciência da Computação
Banca examinadora:
Hendrik Wolter Broer
Ricardo da Silva Torres
Jirí Kosinka
Michael Biehl
Gerard Rudolf Renardel de Lavalette
Jarke Jan van Wijk
Data de defesa: 20-02-2017
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação



Paulo Eduardo Rauber

Visual Analytics Applied to Image Analysis

Análise Visual Aplicada à Análise de Imagens

Banca Examinadora:

- Prof. Dr. Henk Wolter Broer
  Universidade de Groningen
- Prof. Dr. Ricardo da Silva Torres
  Universidade Estadual de Campinas
- Prof. Dr. Michael Biehl
  Universidade de Groningen
- Prof. Dr. Jiří Kosinka
  Universidade de Groningen
- Prof. Dr. Gerard Rudolf Renardel de Lavalette
  Universidade de Groningen
- Prof. Dr. Jarke Jan van Wijk
  Universidade Tecnológica de Eindhoven

A ata da defesa com as respectivas assinaturas dos membros da banca encontra-se no processo de vida acadêmica do aluno.

Campinas, 20 de fevereiro de 2017

Acknowledgments

I would sincerely like to thank the following individuals.

My supervisors, for their unwavering dedication to our work. When we started, I certainly did not imagine how many responsibilities you have, and how hard you must work to achieve your level of success. I wish you even more success in the future.

The assessment committee, for accepting the often thankless task of reviewing a thesis. I hope you found our work inspiring, or at least interesting.

My colleagues at the University of Campinas and at the University of Groningen, for their companionship, despite all the typical challenges faced by graduate students.

My friends, old and new, for all the shared experiences. Specially the laughs.

My mother and the rest of my family. I will always be grateful for your support.

I would also like to thank CAPES (Coordination for the Improvement of Higher Education Personnel), FAPESP (São Paulo Research Foundation, process 2012/24121-9), and the Ubbo Emmius Fund (University of Groningen) for the financial support.

Resumo

Análise de imagens é o campo de pesquisa preocupado com a extração de informações a partir de imagens. Esse campo é bastante importante para aplicações científicas e comerciais.

O objetivo principal do trabalho apresentado nesta tese é permitir interatividade com o usuário durante várias tarefas relacionadas à análise de imagens: segmentação, seleção de atributos, e classificação.

Neste contexto, permitir interatividade com o usuário significa prover mecanismos que tornem possível que humanos auxiliem computadores em tarefas que são de difícil automação.

Com respeito à segmentação de imagens, propomos uma nova técnica interativa que combina *superpixels* com a transformada imagem-floresta. A vantagem principal dessa técnica é permitir rápida segmentação interativa de imagens grandes, além de permitir extração de características potencialmente mais ricas. Os experimentos sugerem que nossa técnica é tão eficaz quanto a alternativa baseada em *pixels*.

No contexto de seleção de atributos e classificação, propomos um novo sistema de visualização interativa que combina exploração do espaço de atributos (baseada em redução de dimensionalidade) com avaliação automática de atributos. Esse sistema tem como objetivo revelar informações que levem ao desenvolvimento de conjuntos de atributos eficazes para classificação de imagens. O mesmo sistema também pode ser aplicado para seleção de atributos para segmentação de imagens e para classificação de padrões, apesar dessas tarefas não serem nosso foco. Apresentamos casos de uso que mostram como esse sistema pode prover certos tipos de informação qualitativa sobre sistemas de classificação de imagens que seriam difíceis de obter por outros métodos.

Também mostramos como o sistema interativo proposto pode ser adaptado para a exploração de resultados computacionais intermediários de redes neurais artificiais. Essas redes atualmente alcançam resultados no estado da arte em muitas aplicações de classificação de imagens. Através de casos de uso envolvendo conjuntos de dados de referência, mostramos que nosso sistema pode prover informações sobre como uma rede opera que levam a melhorias em sistemas de classificação.

Já que os parâmetros de uma rede neural artificial são tipicamente adaptados iterativamente, a visualização de seus resultados intermediários pode ser vista como uma tarefa dependente de tempo. Com base nessa perspectiva, propomos uma nova técnica de redução de dimensionalidade dependente de tempo que permite a redução de mudanças desnecessárias nos resultados causadas por pequenas mudanças nos dados. Experimentos preliminares mostram que essa técnica é eficaz em manter a coerência temporal desejada.

Abstract

We define image analysis as the field of study concerned with extracting information from images. This field is immensely important for commercial and interdisciplinary applications.

The overarching goal behind the work presented in this thesis is enabling user interaction during several tasks related to image analysis: image segmentation, feature selection, and image classification.

In this context, enabling user interaction refers to providing mechanisms that allow humans to assist machines in tasks that are difficult to automate. Such tasks are very common in image analysis.

Concerning image segmentation, we propose a new interactive technique that combines superpixels with the image foresting transform. The main advantage of our proposed technique is enabling faster interactive segmentation of large images, although it also enables potentially richer feature extraction. Our experiments show that our technique is at least as effective as its pixel-based counterpart.

In the context of feature selection and image classification, we propose a new interactive visualization system that combines feature space exploration (based on dimensionality reduction) with automatic feature scoring. This visualization system aims to provide insights that lead to the development of effective feature sets for image classification. The same system can also be applied to select features for image segmentation and (general) pattern classification, although these tasks are not our focus. We present use cases that show how this system may provide a kind of qualitative feedback about image classification systems that would be very difficult to obtain by other (non-visual) means.

We also show how our proposed interactive visualization system can be adapted to explore intermediary computational results of artificial neural networks. Such networks currently achieve state-of-the-art results in many image classification applications. Through use cases involving traditional benchmark datasets, we show that our system may enable insights about how a network operates that lead to improvements along the classification pipeline.

Because the parameters of an artificial neural network are typically adapted iteratively, visualizing its intermediary computational results can be seen as a time-dependent task. Motivated by this, we propose a new time-dependent dimensionality reduction technique that enables the reduction of apparently unnecessary changes in results due to small changes in the data (temporal coherence). Preliminary experiments show that this technique is effective in enforcing temporal coherence.

Samenvatting

Wij definiëren beeldanalyse als het extraheren van informatie uit beelden. Dit veld is extreem belangrijk voor een scala aan commerciële en interdisciplinaire toepassingen.

Gegeven dit veld, het hoofddoel van het werk in dit proefschrift dekt de toepassing van gebruikersinteractie tijdens verschillende beeldanalyse taken zoals beeldsegmentatie, featureselectie en beeldclassificatie. In dit context refereert gebruikersinteractie naar mechanismes die mensen stelt in staat om machines te dienen voor moeilijk automatiseerbare taken, zoals het vaak gebeurt in beeldanalyse.

Voor beeldsegmentatie stellen wij voor een nieuwe interactieve techniek die *superpixels* combineert met de zogenaamde *image foresting transform*. Het voordeel van onze techniek is dat het segmenteren van grote beelden sneller gaat, met als extra waarde het extraheren van potentieel rijkere *features*. Onze experimenten tonen dat onze techniek tenminste zo efficient is als haar pixel-gebaseerde alternatief.

Voor featureselectie en beeldclassificatie stellen wij een nieuwe interactieve visualisatiemethode voor die de exploratie van featurespaces, gebaseerd op dimensionaliteitsreductie, combineert met automatische feature-scoring. Ons systeem verstrekt inzichten die de ontwikkeling van effectieve featureverzamelingen voor beeldclassificatie mogelijk maakt. Naarnaast kan ons systeem toegepast worden om featureselectie te doen voor beeldsegmentatie en algemene patroonclassificatie. Wij demonstreren ons systeem door middel van *use-cases* die laten zien hoe kwalitatief feedback over beeldclassificatie te verkrijgen is dat zeer moeilijk te verkrijgen is via andere (non-visuele) middelen.

Ons voorstel tot interactieve visuele exploratie kan ook aangepast worden om intermediaire rekenresultaten van artificiële neuraalnetwerken te ontdekken, die *state-of-the-art* resultaten bereiken ten opzichte van beeldclassificatie. Wij gebruiken traditionele benchmarks om te laten zien hoe ons systeem inzichten verzamelt over de operatie van een dergelijk netwerk, die leiden tot verbeteringen van een classificatiesysteem.

Gegeven dat de parameters van een artificieel neuraalnetwerk typisch aangepast worden op een iteratieve wijze kan het visualiseren van dergelijke intermediaire resultaten gezien worden als een tijdsafhankelijk proces. Dit leidt ons tot het voorstellen van een nieuwe dimensionaliteitsreductietechniek voor tijdsafhankelijke datasets die onnodige veranderingen in haar resultaten, veroorzaakt door kleine veranderingen in haar inputs, minimaliseert (temporale coherentie). Experimenten laten zien hoe deze techniek effectief is in het creëren van temporale coherentie voor dergelijke multidimensionale datasets.

Contents

Chapter 1

Introduction

We define image analysis as the field of study concerned with extracting information from images. According to this definition, image analysis is highly related to other large fields of study, such as image processing [53], computer vision [140], and pattern recognition [108].

Image analysis is immensely important for commercial and interdisciplinary applications [53, 31]. Physics, engineering, and biology have all benefited from its advancements.

In the last decades, significant advancements have been made in several image analysis tasks, such as image segmentation (Sec. 1.1) and image classification (Sec. 1.2). Despite these advancements, humans still outperform machines in many seemingly simple tasks that require high flexibility [140]. In general, if intelligence is defined as a measure of the ability of an agent to achieve goals in a wide range of environments [89], then cooperation between machines and users will remain relevant for as long as machines lack human-level intelligence.

The overarching goal behind the work presented in this thesis is enabling user interaction in image analysis tasks. In this context, enabling user interaction refers to providing mechanisms that allow humans to assist machines in tasks that are difficult to fully automate.

In particular, we advocate the use of visual analytics to incorporate user expertise into the design and operation of image analysis methods. In simple terms, visual analytics is a field of study concerned with the application of interactive visualization techniques to assist in data analysis [78]. This field has recently been advocated as an effective solution for analyzing *black-box* methods [107], which include many methods employed in image analysis.

This thesis focuses mainly on three tasks in image analysis: image segmentation, feature selection, and image classification. Such focus is justified by the importance of these tasks, and relatively high complexity of the corresponding solutions. However, it should be clear that visual analytics is not limited to these tasks.

This chapter is organized as follows. Sections 1.1, 1.2, and 1.3 introduce the tasks that we address using visual analytics. Section 1.4 describes the task of visualizing time-dependent data, which is highly related to the task introduced in Sec. 1.3. Section 1.5 presents the main research question that unifies our work. Finally, Section 1.6 describes how this thesis is organized, and details our contributions towards the tasks introduced

in the previous sections.

1.1 Image segmentation

In simple terms, image segmentation is the task of partitioning an image into objects of interest [53]. This task may be an ultimate goal (*e.g.*, segmenting a person to be copied onto another image), or part of a larger image analysis pipeline (*e.g.*, segmenting a skin lesion to be automatically classified as benign or malignant [73]).

Image segmentation is extremely difficult to automate [53], particularly because objects of interest are generally ill-defined. This leads to an ideal application for interactive methods.

In interactive image segmentation, the user indicates the approximate localization of an object of interest (*e.g.*, by placing markers), while the machine performs careful delineation of this object based on image characteristics or pre-defined models [43, 41, 19, 120, 32]. Such delineation is typically the most error-prone and time-consuming step in manual segmentation.

As we detail in Sec. 1.6, in this context, we are interested in enabling effective and efficient interactive segmentation.

1.2 Image classification and feature selection

Image classification is the task of assigning a class label to an image based on generalization from examples (available in a so-called training set). The typical solution to this task involves representing each image by an observation (real vector), whose features (elements) correspond to measured characteristics related to colors, textures, and shapes [33].

Feature selection is crucial for effective image classification. In broad terms, feature selection is the task of finding a subset of candidate features that is small and sufficient for a particular purpose. While using too few features can lead to poor generalization, using too many features can be prohibitively expensive to compute, or even introduce confounding information into the training data [60, 93].

Human experts generally evaluate choices involved in designing classification systems using cross-validation [108]. However, this approach is typically limited by the feedback that numeric classification efficacy measures can provide. As a consequence, when suboptimal results are obtained, these experts are often left unaware of which aspects limit classification system efficacy, and what can be done to improve these systems. This and other issues have been referred to as the "black art" of machine learning [35].

As we detail in Sec. 1.6, in this context, we are interested in enabling interactive feature selection for image classification system design, and, most importantly, providing insights that lead to the improvement of such systems.

1.3 Image classification by artificial neural networks

Although the traditional features mentioned in the previous section are still widely employed in image classification, particularly when large training sets are unavailable, artificial neural networks recently became able to achieve extraordinary results in raw (or pre-processed) image classification, bypassing traditional feature selection [130].

However, choosing appropriate pre-processing steps, architectures, and hyperparameters for these networks is a challenging task. This is confirmed by the vast literature on the subject [113, 12], and the widespread use of heuristics that are poorly understood according to their own proponents (*e.g.*, DropConnect [152]).

The difficulty in designing effective artificial neural networks is arguably related to the difficulty in interpreting how such networks arrive at decisions for a given input image. As an extreme example, it is easy to create images of unrecognizable objects that a particular network classifies with absolute certainty [111].

As we detail in Sec. 1.6, in this context, we are interested in enabling the inspection of artificial neural networks using visual analytics, and, most importantly, once again providing insights that lead to improvements along the image classification pipeline.

1.4 Time-dependent data visualization

Our visual analytics approach towards the tasks described in Secs. 1.2 and 1.3 is highly dependent on visualizing datasets composed of high-dimensional vectors (observations).

Our approach also depends on representing such a dataset, seen as a sequence of observations, by a two-dimensional projection obtained through dimensionality reduction. In this context, we define a two-dimensional projection as a sequence of points in the plane, which correspond to the observations in a dataset. Although dimensionality reduction techniques vary in specific goals, they all attempt to represent some aspect of the so-called structure of a dataset in its projection. This structure is characterized by distances between observations, presence of clusters, and overall spatial data distribution [95, 88]. In comparison to other high-dimensional data visualization alternatives, dimensionality reduction is remarkably scalable with respect to the number of observations and features.

As will become clear, because the parameters of an artificial neural network are typically adapted iteratively, our visual analytics approach towards these models becomes time-dependent. Such time-dependent process can be represented by a sequence of datasets, where each dataset corresponds to a particular time step, and each observation has a corresponding observation across time.

Unfortunately, visualizing a sequence of datasets using traditional dimensionality reduction techniques is not always straightforward. For instance, although it is possible to create a projection independently for each dataset in such a sequence, this strategy (and similar alternatives) often leads to temporal incoherence: significant variability in the resulting sequence of projections that does not reflect significant variability in the sequence of datasets. This is mostly due to the fact that many dimensionality reduction techniques are highly sensitive to small changes in their inputs [49]. This issue affects a state-of-the-art dimensionality reduction technique called t-SNE [148], which is widely

employed in our work. Therefore, temporal incoherence compromises one of our fundamental contributions by potentially leading to incorrect insights.

As we detail in Sec. 1.6, in this context, we are interested in enabling the reliable visualization of time-dependent high-dimensional data.

1.5 Research question

Considering the tasks outlined in Secs. 1.1-1.4, we can state our central research question as follows:

How can we provide actionable insights about the design and operation of image analysis methods through visual analytics?

In this context, actionable insight refers to information that enables a user to improve a process.

The next section describes how this thesis is organized, and details contributions that aim to address our central research question.

1.6 Thesis structure

This thesis is organized as follows.

Chapter 2 provides an overview of related work in image segmentation, pattern classification, feature selection, and high-dimensional data visualization. This overview is complemented by more specific descriptions of related work within each following chapter.

Chapter 2 also thoroughly details the techniques that are employed in our work. With exceptions that are clearly mentioned, the reader is welcome to skip the highly technical sections, and refer back on the basis of necessity, since the following chapters are mostly self-contained in a higher abstraction level.

As an implicit goal, Chapter 2 shows that many widely used techniques demand non-trivial choices that require significant expertise, which is important to motivate our application of visual analytics.

Chapter 3 proposes a new interactive segmentation technique based on the successful image foresting transform (IFT) [42]. The IFT algorithm can be applied to segment multiple objects of interest in linear or linearithmic time on the number of pixels in an image. Notably, segmenting multiple objects simultaneously is NP-hard using other widespread segmentation approaches that find optimum cuts in graphs [26, 81].

Our contribution to IFT-based interactive segmentation is the introduction of superpixels. A superpixel is typically a small region within an image, whose pixels are uniform with respect to some criteria (*e.g.*, color, texture). In this context, the main benefit of using superpixels (as atomic units instead of pixels) is reducing the computational time required by interactive segmentation. Although superpixels require significant upfront computational time, which can be allocated before user involvement, such cost is amortized during interactive segmentation. Our experiments also suggest that superpixels are capable of achieving at least comparable efficacy to pixels.

Another potential advantage of superpixels is enabling richer feature extraction [143, 155, 100]. In this context, feature extraction refers to the task of representing each superpixel by an observation (real vector). Each element in such an observation corresponds to a feature, and represents a characteristic measured from the corresponding superpixel (*e.g.*, mean red color component). Because our segmentation approach relies on distances between observations, the choice of features is crucial to its success. We return to the problem of feature selection in the context of image classification in Ch. 4.

Conducting significant user studies to evaluate interactive methods is generally a difficult and costly process. However, the evaluation of interactive image segmentation methods is a very particular exception, because automated experimentation is straightforward. Concretely, consider a set of images whose segmentation (ground truth) is known. In this case, it is possible to simulate interactive segmentation by having the machine act as a so-called *robot user*. In Chapter 3, we also propose novel robot users that employ distinctive strategies to evaluate interactive segmentation methods.

Chapter 4 proposes a system that enables interactive feature selection for image classification. The system integrates feature space exploration with automatic feature evaluation, and attempts to provide insights that lead to the development of effective feature sets. The same system can also be employed to select features for other tasks, such as image segmentation or (general) pattern classification.

The proposed interactive system employs dimensionality reduction to visually represent a dataset (sequence of observations) by a two-dimensional projection. As we already mentioned, a projection attempts to represent some aspect of the so-called structure of a dataset, and is remarkably scalable with respect to the number of observations and features.

The experiments presented in Chapter 4 indicate that projections may provide a kind of qualitative feedback about classification systems that would be very difficult to obtain by other (non-visual) means. Although there is no guarantee that a projection will provide insightful feedback about a particular dataset, as we also exemplify in Ch. 4, the proposed approach requires only a small upfront user effort investment.

Chapter 5 shows how the approach originally proposed to visualize inputs to an image classification system can be adapted to visualize intermediary computational results of an artificial neural network. Fortunately, the computations carried by artificial neural networks produce convenient intermediary results. Concretely, each layer of artificial neurons transforms an input vector (or image) into an output vector (or image). The parameters that control these transformations are adjusted to obtain correct classifications in a training set. Therefore, in broad terms, the output of a given layer for an input image can be interpreted as an alternative representation learned by a network. Since these learned representations (also called activations) can be represented by real vectors, they can be explored using the approach proposed in Chapter 4.

Chapter 5 also presents experiments performed in established artificial neural network benchmark datasets. The results indicate that our interactive approach may enable insights about how a network operates that lead to improvements along the classification pipeline.

Chapter 6 describes a new time-dependent dimensionality reduction technique that

allows a controllable trade-off between temporal coherence and spatial coherence (defined as preservation of structure at a particular time step).

As we alluded to in Sec. 1.4, in a broad sense, the activations of an artificial neural network for a given a set of inputs represent a particular stage of a time-dependent process, since the parameters of a network are typically adapted iteratively. This time-dependent process can be represented by a sequence of datasets, such that each dataset corresponds to a particular time step, and each observation (activation) has a corresponding observation (activation) across time. As we also mentioned in Sec. 1.4, visualizing a sequence of datasets using traditional dimensionality reduction techniques may lead to temporal incoherence: significant variability in the resulting sequence of projections that does not reflect significant variability in the sequence of datasets.

Therefore, although temporal incoherence is not strictly related to visual analytics, it impairs our proposed visual analytics approach to explore artificial neural networks applied to image classification.

The preliminary experiments presented in Chapter 6 suggest that our new technique is successful in enforcing temporal coherence and encouraging smooth changes between projections.

Chapter 7 summarizes our contributions and suggests future work.

Chapter 2

Related work

This chapter summarizes related work, and organizes it into four categories: image segmentation (Sec. 2.2), pattern classification (Sec. 2.3), feature selection (Sec. 2.4), and high-dimensional data visualization (Secs. 2.5 and 2.6). This chapter also serves another important purpose: describing in detail the techniques that are used to a significant extent in the next chapters, or that are highly related to the tasks that we address. Understanding some of these techniques is not strictly required for reading the next chapters, which are mostly self-contained. Therefore, the reader is encouraged to refer back to the technical sections in this chapter based on demand. Our goal is to provide a complete and contextualized description that allows discussing the assumptions underlying the well-known techniques. Such description also allows a concrete presentation of the lesser known techniques, which often requires the provided background. An additional goal of this chapter is to show that many widely used techniques demand non-trivial choices that require significant expertise, which will help motivate our application of visual analytics.

2.1 Preliminaries

In this text, we distinguish between real vectors and real column matrices. We denote vectors by lower case bold letters, and matrices by upper case bold letters. Exceptionally, we let $\mathbf{x}^T$ denote the row matrix corresponding to the vector $\mathbf{x}$.

The inner product between vectors $\mathbf{u}$ and $\mathbf{v}$ is denoted by $\mathbf{uv} = \sum_i u_i v_i$, where u_i is the i-th element of vector $\mathbf{u}$. The typical operations between matrices and vectors (addition, multiplication) treat vectors as if they were column matrices, and result in matrices.

The gradient function $\nabla f : \mathbb{R}^D \to \mathbb{R}^D$ of a differentiable function $f : \mathbb{R}^D \to \mathbb{R}$ of multiple variables $x_1, \ldots, x_D$ is given by

$$\nabla f(\mathbf{a}) = \Big(\frac{\partial}{\partial x_1} f(\mathbf{a}), \ldots, \frac{\partial}{\partial x_D} f(\mathbf{a}) \Big), \tag{2.1}$$

for every $\mathbf{a} \in \mathbb{R}^D$. When ambiguity is impossible, we also let $\nabla f(\mathbf{x})$ denote the gradient of f at point $\mathbf{x} = (x_1, \ldots, x_D)$, which overloads variable names with their corresponding values.

We denote random variables by upper case letters, and random vectors by upper case

bold letters. We let $X \sim p_X$ denote that the random variable X is distributed according to the probability density (or mass) function p_X. The set of valid assignments to the random variable X is denoted by $\text{Val}(X)$. The expected value of random variable X is a scalar denoted by $\mathbb{E}[X]$, and its standard deviation is the scalar $\text{std}[X]$. The expected value of a random vector $\mathbf{X} = (X_1, \ldots, X_D)$ is a vector denoted by $\mathbb{E}[\mathbf{X}]$, whose elements correspond to the expected value of each random variable in $\mathbf{X}$.

The covariance $\text{cov}[X, Y]$ between random variables X and Y is given by $\text{cov}[X, Y] = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$. The (Pearson) correlation coefficient $\text{corr}[X, Y]$ between random variables X and Y is given by

$$\text{corr}[X, Y] = \frac{\text{cov}[X, Y]}{\text{std}[X]\,\text{std}[Y]}, \tag{2.2}$$

when $\text{std}[X]\,\text{std}[Y] > 0$, and is always between -1 and 1. A high correlation coefficient (in absolute value) indicates a highly linear relationship between X and Y, and the sign indicates whether the variables are proportional or inversely proportional. In particular, $|\,\text{corr}[X, Y]| = 1$ implies $Y = aX + b$ for some $a, b \in \mathbb{R}$, where $a \neq 0$. If X is independent of Y, denoted by $X \perp\!\!\!\perp Y$, then $\text{corr}[X, Y] = 0$. However, the converse is not generally true, and therefore the correlation coefficient is not an appropriate measure of dependence between random variables [108].

The covariance matrix of a random vector $\mathbf{X} = (X_1, \ldots, X_D)$ is given by $\text{cov}[\mathbf{X}] = \mathbb{E}[(\mathbf{X} - \mathbb{E}[\mathbf{X}])(\mathbf{X} - \mathbb{E}[\mathbf{X}])^T]$, where $\mathbf{Z}^T$ denotes the random row matrix corresponding to a random vector $\mathbf{Z}$. In other words, $\text{cov}[\mathbf{X}]_{i,j} = \text{cov}[X_i, X_j]$. The correlation matrix is analogous.

We will employ a widespread abuse of notation that vastly simplifies the presentation in the next sections [108, 15]: probability functions will sometimes omit the subscripts that relate them to variables. For instance, the probability density associated to the joint assignment $X = x$ and $Y = y$ given $Z = z$ will be written as either $p_{X,Y|Z}(x, y, z)$ or $p(x, y \mid z)$. This implies that the same letter may denote different probability functions depending on arguments, although we are always careful to avoid excessive ambiguity.

The concept of an independent, identically distributed (*iid*) dataset will be useful in the next sections, and is reviewed next. It is worth noting that we represent unknown parameters using random variables, which is typical in Bayesian statistics [81, 108, 15].

Consider a non-empty sequence of random vectors $\boldsymbol{\mathcal{D}} = \mathbf{X}_1, \ldots, \mathbf{X}_N$, and a random (parameter) vector $\boldsymbol{\Theta}$. Let $\mathbf{X}_{-i}$ denote the set of all random vectors in $\boldsymbol{\mathcal{D}}$ excluding random vector $\mathbf{X}_i$. Furthermore, suppose that each of the random vectors in the sequence $\boldsymbol{\mathcal{D}}$ is distributed according to the same probability function, which is conditional on a particular assignment to $\boldsymbol{\Theta}$. Concretely, if we let $\boldsymbol{\theta}^* \in \text{Val}(\boldsymbol{\Theta})$ denote the (possibly unknown) true parameter vector, then $\mathbf{X}_i \sim p(\cdot \mid \boldsymbol{\theta}^*)$, for all i, for some conditional probability function $p(\cdot \mid \boldsymbol{\theta}^*)$. If $\mathbf{X}_i \perp\!\!\!\perp \mathbf{X}_{-i} \mid \boldsymbol{\Theta}$ for all i, then $\boldsymbol{\mathcal{D}}$ is an independent, identically distributed sample according to $p(\cdot \mid \boldsymbol{\theta}^*)$.

Intuitively, a sample $\boldsymbol{\mathcal{D}}$ is iid if any parameter vector $\boldsymbol{\theta}$ determines the probability function associated to each random vector in $\boldsymbol{\mathcal{D}}$ completely, such that knowledge about the other random vectors becomes irrelevant.

A dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ is iid according to $p(\cdot \mid \boldsymbol{\theta}^*)$ if it corresponds to a sequence

of assignments to the sequence of random vectors in the sample $\boldsymbol{\mathcal{D}}$, as defined above. For our purposes, the likelihood of the dataset $\mathcal{D}$ given the parameter $\boldsymbol{\theta}$ (traditionally called the likelihood of the parameter $\boldsymbol{\theta}$) is defined as the probability/density $p(\mathcal{D} \mid \boldsymbol{\theta})$ of the dataset $\mathcal{D}$ given $\boldsymbol{\theta}$. In other words, the likelihood is the probability/density of the dataset $\mathcal{D}$ assuming that $\boldsymbol{\theta}^* = \boldsymbol{\theta}$, which is given by

$$p(\mathcal{D} \mid \boldsymbol{\theta}) = \prod_{i=1}^{N} p(\mathbf{x}_i \mid \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(\mathbf{x}_i \mid \boldsymbol{\theta}), \tag{2.3}$$

where the first equality follows from the chain rule of probability (with no further assumptions), and the second from our assumption that $\mathbf{X}_i \perp\!\!\!\perp \mathbf{X}_{-i} \mid \boldsymbol{\Theta}$, for all i. Notice the abuse of notation, since p represents distinct probability functions in the equation above (except if $N = 1$).

Consider an iid dataset $\mathcal{D} = x_1, \ldots, x_N$ distributed according to p. If $X \sim p$ and its expected value $\mathbb{E}[X]$ exists and is finite, the law of large numbers [153] guarantees that $N^{-1} \sum_i x_i \to \mathbb{E}[X]$ as $N \to \infty$. In other words, the mean of a (very large) iid dataset is probably a good approximation to the expected value of X. This is a particular case of Monte Carlo approximation, which can be used to approximate the expected value of any function of random variables (which is always itself a random variable).

For instance, if $\bar{x}$ is a Monte Carlo approximation of $\mathbb{E}[X]$ considering the dataset $\mathcal{D}$, the variance $\text{var}[X]$ of X can be approximated by $N^{-1} \sum_i (x_i - \bar{x})^2$. It can be shown that this approximation underestimates $\text{var}[X]$ on average (in a very precise sense, particularly when N is small), precisely because it also requires an estimate of $\mathbb{E}[X]$ [15, 153]. It can also be shown that this bias is corrected through multiplying the approximation by $\frac{N}{N-1}$.

Considering an iid dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ distributed according to p, a Monte Carlo approximation to the covariance matrix is given by

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T, \tag{2.4}$$

where $\bar{\mathbf{x}}$ is the empirical mean (Monte Carlo approximation of the expected value of a random vector $\mathbf{X} \sim p$). The bias in this approximation is also corrected through multiplication by $\frac{N}{N-1}$ [153].

2.2 Image segmentation

Image segmentation is the task of partitioning an image into objects of interest (see Fig. 2.1). The concept of object of interest is highly dependent on context. Precisely for this reason, user interaction still is essential for effective segmentation.

Image segmentation can be further divided into two tasks: recognition and delineation [44]. Recognition corresponds to establishing the approximate localization of the objects of interest, while delineation corresponds to discovering precisely which pixels belong to each of these objects.

Since recognition is very dependent on context, humans usually outperform machines,

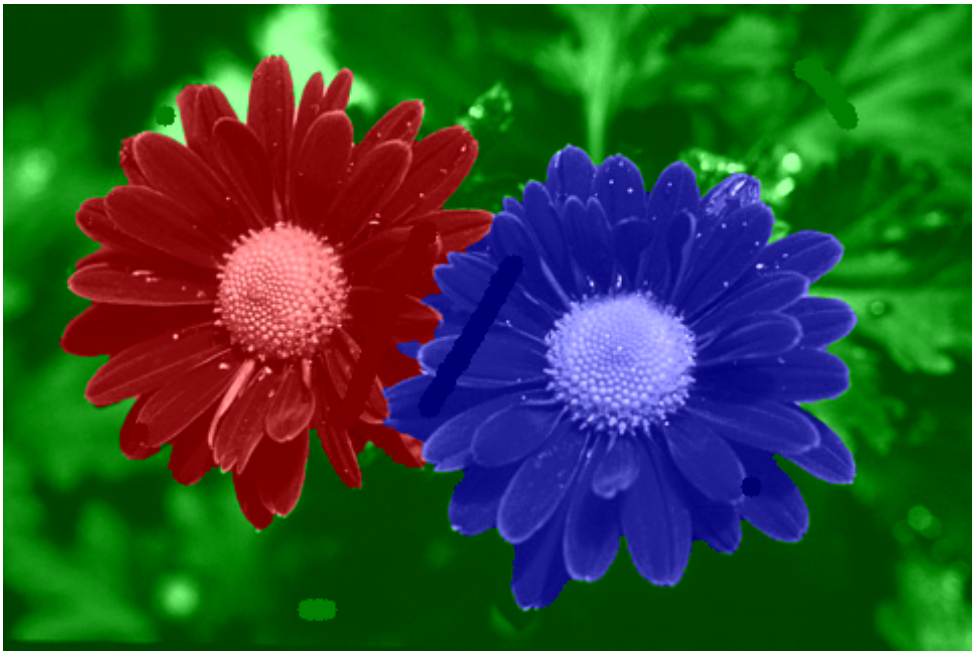


Figure 2.1: Segmented image. Different hues represent different segments.

but the latter have great potential for the minutiae involved in delineation. Accordingly, several approaches have explored a combination of user and machine effort [43, 41, 19, 120, 32]. A common strategy in several of these approaches is called operator-assisted synergistic segmentation. This strategy consists in the creation of *seeds* (sets of pixels corresponding to the same object of interest) by the user, and automatic delineation by the machine. Such delineation can be corrected interactively with the addition (or removal) of seeds. Thus, the final accuracy depends on the quality of the delineation and on the effort devoted by the user to the task.

It is useful to categorize segmentation methods into three groups [26]: purely image-based, appearance model-based, and hybrid.

Purely image-based: These methods delineate objects based on information that can be entirely obtained from the image and/or user input. Such methods include level sets, active boundaries, fuzzy connectedness, graph cuts, watersheds, clustering and Markov random fields [26]. Perhaps the most widespread family of methods are graph cuts based on the maximum flow algorithm. However, this algorithm has severe limitations. For instance, the cost function minimized by some graph cut variants tends to favor smaller boundaries [26]. More importantly, simultaneously segmenting more than two objects using graph cut methods is an NP-hard problem [26, 81]. While it is possible to segment each object individually, merging the resulting segmentation is not trivial. In contrast, segmentation algorithms based on the image foresting transform (IFT) are capable of segmenting multiple objects in linear time, and do not favor smaller boundaries [42, 26]. This transform is a tool for the design of operators based on optimum connectivity, and has been successfully applied to the development of algorithms for image processing [42], pattern classification [115], data clustering [127], and active learning [129]. It is interesting to note that fuzzy connectedness methods, which can be efficiently implemented using the

IFT algorithm, also define an optimum cut in a graph given a particular energy function [26]. We will present the image foresting transform together with a new segmentation approach in Chapter 3.

Model-based: These methods use information about objects encoded into models to perform the segmentation. These methods include active shape models [28] and atlas-based models [48], which have been applied to segmentation of anatomic structures of the brain given magnetic resonance images [37, 56].

Hybrid: As the name implies, hybrid methods combine these two previous approaches, attempting to overcome their individual weaknesses [94]. The clouds model [104] is an example of such approach, which was applied to segment magnetic resonance images. Both model-based and hybrid methods, however, are less generally applicable than the purely image-based segmentation methods, which are most benefited by user interaction.

2.3 Pattern classification

In *supervised learning*, a subfield of machine learning, the important task of pattern classification consists on assigning a class label to a high-dimensional vector based on generalization from previous examples [108]. In broad terms, this task is typically solved by finding parameters for a classification model that maximize a measure of efficacy.

More concretely, consider an iid dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$. Every pair $(\mathbf{x}_i, y_i) \in \mathcal{D}$ is composed of an *observation* $\mathbf{x}_i \in \mathbb{R}^D$, and a *class* $y_i \in \{1, \ldots, C\}$, where C is the number of *classes*. The j-th element of $\mathbf{x}_i$ corresponds to *feature* j. For instance, the observations may correspond to images of animals, and the classes to the C distinct species present in the images.

The typical goal is to find a classifier $f : \mathbb{R}^D \to \{1, \ldots, C\}$ that maps observations to classes based on the (training) set $\mathcal{D}$, and *generalizes* well to new observations. Although generalization can be defined precisely in the context of Bayesian decision theory [108], it is typically evaluated by the efficacy on a so-called *test set*. A test set $\mathcal{D}'$ is a dataset that was not considered to find the classifier f, and is iid according to the same distribution as the training set $\mathcal{D}$.

A common measure of efficacy is the accuracy of the classifier f on the test set $\mathcal{D}'$, which is given by

$$\frac{1}{N'} \sum_{(\mathbf{x},y)\in\mathcal{D}'} \mathbb{I}(f(\mathbf{x}) = y), \tag{2.5}$$

where N' is the number of observations in $\mathcal{D}'$, and $\mathbb{I}$ is the indicator function, which is 1 if its argument is true, and 0 otherwise. In words, the accuracy is the fraction of correct *classifications* on the test set.

Alternatively, it is also possible to model the conditional probability $p(y \mid \mathbf{x})$ of class y given the observation $\mathbf{x}$, for every $(\mathbf{x}, y) \in \mathbb{R}^D \times \{1, \ldots, C\}$. The advantage is that this approach provides a measure of uncertainty about classifications [15, 108]. The corresponding classifier f may be given by $f(\mathbf{x}) = \arg\max_y p(y \mid \mathbf{x})$.

Pattern classification is a challenging task, partly due to its extremely large design space. For our purposes, this task can be divided into representation and learning, as

follows.

The representation task is concerned with how objects of interest are modeled as observations. In general, elements of these vectors correspond to measurable characteristics (features) of the objects. Usually, many different features can be considered, and it is unclear which of them are valuable for generalization. For example, consider the task of image classification. A wide variety of color, texture, shape, and local features can be extracted from images [33]. Using too few features can lead to poor generalization; while using too many features can be prohibitively expensive to compute, or even introduce confounding information into the training data [60, 93]. Deep neural networks recently became able to bypass feature design by dealing directly with raw images [84, 11]. However, such networks require very large amounts of labeled (training) data, which are not always available, and pose additional design challenges of their own [12]. Therefore, feature selection for classification system design still is a very important problem. We return to this issue in Section 2.4.

Once a representation is available, the learning task consists on selecting, applying, fine-tuning, and testing learning algorithms. A huge number of such algorithms exists, based on a wide variety of principles, and no single algorithm is the best for every situation [154]. Learning algorithms such as k-nearest neighbors, naive Bayes, support vector machines, decision trees, artificial neural networks, and their ensembles, have been applied in a wide variety of practical problems [108].

Since the objective of pattern classifiers is to generalize from previous experience, hyperparameter search and efficacy estimation are usually performed using cross-validation [108, 79], which we introduce in Sec. 2.3.1. However, this approach is bounded by the limited feedback that numerical (classification) efficacy measures can provide. As a consequence, when suboptimal results are obtained, designers are often left unaware of which aspects limit classification system efficacy, and what can be done to improve these systems. This and other issues have been referred to as the "black art" of machine learning [35].

Diagnosing the cause of poor generalization in classification systems is a hard problem. Options include using cross-validation to compute efficacy indicators (*e.g.*, accuracy, precision and recall, area under the ROC curve), and learning curves, which show generalization performance for an increasing training set. In multi-class problems, confusion matrices can also be used to diagnose mistakes between classes [45]. Information visualization systems can also be helpful in this diagnostic process, as we discuss in Chapter 4.

The following sections describe five supervised learning techniques in detail: k-nearest neighbors (KNN), logistic regression (LR), support vector machines (SVM), decision trees (DT, including random forest classifiers and extremely randomized trees), and artificial neural networks (ANN, including multilayer perceptrons and convolutional neural networks). Some of these techniques will be directly used in Chapter 4 (KNN, SVM, DT), while others are pre-requisites for understanding feature selection techniques that we employ (LR, SVM, DT). Artificial neural networks are the main subject of Chapter 5.

As noted previously, understanding the details behind these techniques is not strictly required for the next chapters, which are self-contained. Therefore, the reader is welcome to the skip technical details, and refer back to these sections whenever necessary. One

exception is Section 2.3.1, which is used to introduce important concepts in supervised learning, including hyperparameters, the curse of dimensionality, and model selection.

2.3.1 K-nearest neighbors

K-nearest neighbors is arguably the simplest widespread classification technique [108]. Given a dataset $\mathcal{D}$, this technique assumes that the probability $p(y \mid \mathbf{x})$ of class y given observation $\mathbf{x}$ is given by

$$p(y \mid \mathbf{x}) = \frac{1}{K} \sum_{(\mathbf{x}_i, y_i) \in N(\mathbf{x}, \mathcal{D}, K)} \mathbb{I}(y_i = y), \tag{2.6}$$

where $\mathbb{I}$ is once again the indicator function, and $N(\mathbf{x}, \mathcal{D}, K)$ is a subset of $\mathcal{D}$ whose observations are the K closest to $\mathbf{x}$ (ties broken arbitrarily), according to a distance function on $\mathbb{R}^D$ (*e.g.*, Euclidean distance). In other words, $p(y \mid \mathbf{x})$ is the fraction of the K nearest neighbors of $\mathbf{x}$ in $\mathcal{D}$ that belong to class y. For an illustrative example, see Fig. 2.2.

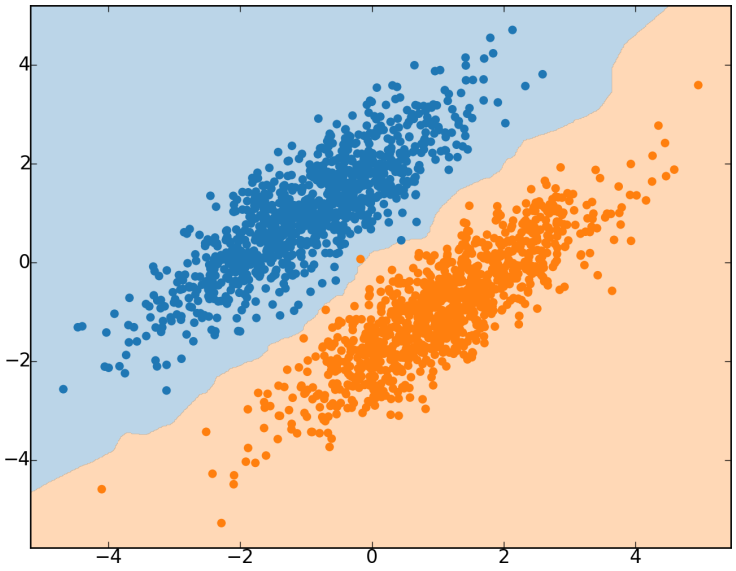


Figure 2.2: K-nearest neighbors ($K = 3$) decision boundary for a 2D dataset. Each point corresponds to an observation, and is colored according to its class. The region in orange/blue contains points that would be assigned to the orange/blue class during testing.

Clearly, a 1-nearest neighbor classifier achieves perfect accuracy on the training set (assuming there are contradictory observation-class pairs). However, this does not imply that the classifier generalizes well. Although K-nearest neighbors can generalize well given an appropriate distance function and sufficient data, it can also suffer from the *curse of dimensionality*, a difficulty that may arise in high-dimensional spaces [15]. As an example of this curse, consider a hypercube with unit volume. A hypercube with side $l \leq 1$ inside such unit hypercube occupies a fraction $F(l) = l^D$ of the unit volume. Thus, to occupy a fraction r of the volume of the unit hypercube, a hypercube must have side $L(r) = \sqrt[D]{r}$.

In the case of $D = 100$, to occupy $r = 1\%$ of the volume of the unit hypercube, the smaller hypercube must have sides larger than 0.95. This example aids the intuition that a technique like K-nearest neighbors may base its decisions on neighbors that are considerably distant in the high dimensional space (depending on the dataset). Other techniques may achieve better generalization by making particular assumptions about the dataset. For the same reason, there is no single *best* learning algorithm for every practical situation [154, 108].

Notice that K-nearest neighbors requires fixing K before the classifier f can be obtained from the training set, which makes K a *hyperparameter*. The general task of choosing hyperparameters that generalize well is called *model selection* [108]. As already mentioned, generalization cannot be evaluated on the training set. Hyperparameters should not be chosen based on performance on the test set either, because doing so would introduce an *optimistic* bias. In short, it would not be clear how hyperparameters chosen for a particular test set would generalize for other test sets.

A typical solution to the problem of model selection is to partition the training set into a validation set and an effective training set. For each choice of hyperparameters, the classifier is fitted to the effective training set, and evaluated in the validation set. This procedure is called cross-validation. In F-fold cross-validation, the training set is partitioned into F subsets (folds). Each fold is used as a validation set while the others are used as effective training sets, and the hyperparameters that achieve best (average) efficacy results over each validation fold are selected. The selected hyperparameters are used to fit a classifier to the whole training set, which can be finally evaluated on the test set.

In summary, K-nearest neighbors is a simple classification technique, which requires choosing a number of neighbors K and an appropriate distance function. This section also introduced important concepts in machine learning: hyperparameters, curse of dimensionality, model selection, and cross-validation.

2.3.2 Logistic regression

Consider a dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, which is iid according to $p(\cdot \mid \boldsymbol{\theta}^*)$ for some unknown parameter vector $\boldsymbol{\theta}^*$. Furthermore, suppose $y_i \in \{0, 1\}$, for all i. Consider the task of binary classification, which corresponds to predicting binary targets from observations based on generalization from $\mathcal{D}$.

Logistic regression [108] is a technique that assumes that the probability $p(y \mid \mathbf{x}, \mathbf{w})$ of class $y \in \{0, 1\}$ given observation $\mathbf{x}$ and weight vector $\mathbf{w}$ is given by

$$p(y \mid \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}\mathbf{x})^y (1 - \sigma(\mathbf{w}\mathbf{x}))^{1-y}, \tag{2.7}$$

where σ is the sigmoid function given by

$$\sigma(t) = \frac{1}{1 + e^{-t}}, \tag{2.8}$$

for all $t \in \mathbb{R}$.

Intuitively, a logistic regression model assigns more probability to class 1 whenever $\mathbf{wx} > 0$, and each element w_j of $\mathbf{w}$ indicates how much feature j contributes (or detracts) to the detection of class 1. Because the probability of the distinct classes given $\mathbf{x}$ is the same if and only if $\mathbf{wx} = 0$, the so-called *decision boundary* of this classification technique is the hyperplane $S = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{wx} = 0\}$. For an illustrative example, see Fig. 2.3.

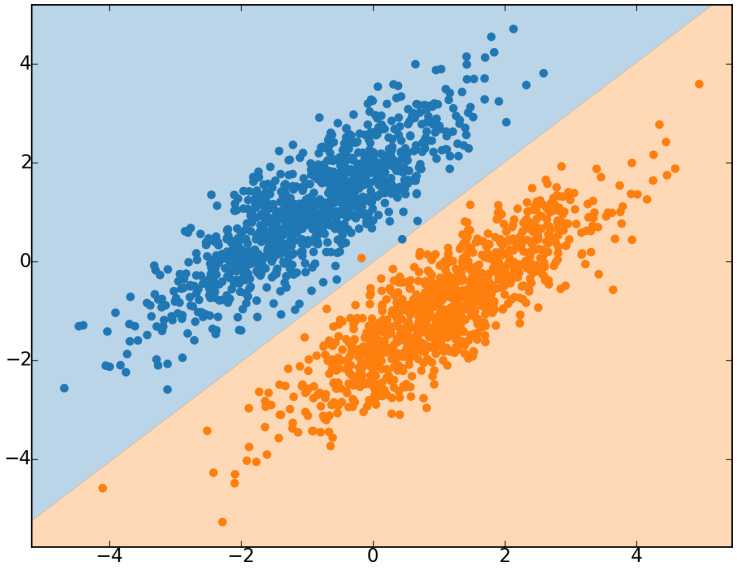


Figure 2.3: Logistic regression ($\lambda = 1$) decision boundary for a 2D dataset.

Although the assumptions made by logistic regression appear very restrictive, an input observation $\mathbf{x}$ can be transformed by a pre-defined feature map $\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^{D'}$ before the technique is applied. In that case, we say the probability $p(y \mid \mathbf{x}, \mathbf{w})$ is given by

$$p(y \mid \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}\boldsymbol{\phi}(\mathbf{x}))^y (1 - \sigma(\mathbf{w}\boldsymbol{\phi}(\mathbf{x})))^{1-y}. \tag{2.9}$$

Therefore, the decision boundary is no longer (necessarily) a hyperplane on the original space.

In what follows, we assume that the observations were already transformed by a feature map. It is advisable to at least prefix each original observation $\mathbf{x}$ by the constant 1, leading to a transformed observation $\mathbf{x}' = (1, \mathbf{x})$. In this case, the decision boundary on the original space becomes an affine hyperplane $S = \{\mathbf{x} \in \mathbb{R}^D \mid w_0 + \mathbf{w}_{1:D}\mathbf{x} = 0\}$.

The (conditional) likelihood $p(\mathcal{D} \mid \mathbf{w})$ of the weight vector $\mathbf{w}$ given the dataset $\mathcal{D}$ is given by

$$p(\mathcal{D} \mid \mathbf{w}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \mathbf{w}) = \prod_{i=1}^{N} \sigma(\mathbf{w}\mathbf{x}_i)^{y_i} (1 - \sigma(\mathbf{w}\mathbf{x}_i))^{1-y_i}. \tag{2.10}$$

Notice that this likelihood is not based on the joint density over observations and classes, which is irrelevant for our purposes. Instead, $p(\mathcal{D} \mid \mathbf{w})$ is the probability associated to the particular assignment $\mathbf{y} = (y_1, \ldots, y_N)$ when the observations $\mathbf{x}_1, \ldots, \mathbf{x}_N$ are seen as constants (according to $\mathcal{D}$), and $Y_i \perp\!\!\!\perp \mathbf{X}_{-i}, Y_{-i} \mid \mathbf{X}_i, \mathbf{W}$.

The log-likelihood $\ell(\mathbf{w}) = \log p(\mathcal{D} \mid \mathbf{w})$ is given by

$$\ell(\mathbf{w}) = \log \prod_{i=1}^{N} \sigma(\mathbf{w}\mathbf{x}_i)^{y_i}(1 - \sigma(\mathbf{w}\mathbf{x}_i))^{1-y_i} \tag{2.11}$$

$$= \sum_{i=1}^{N} \log \left[\sigma(\mathbf{w}\mathbf{x}_i)^{y_i}\right] + \log \left[(1 - \sigma(\mathbf{w}\mathbf{x}_i))^{1-y_i}\right] \tag{2.12}$$

$$= \sum_{i=1}^{N} y_i \log \sigma(\mathbf{w}\mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}\mathbf{x}_i)). \tag{2.13}$$

It can be shown that there is no analytical expression for the maximum (log-)likelihood estimate $\hat{\mathbf{w}}$ for a logistic regression model [108]. However, the log-likelihood ℓ is concave, and so there is at most one local (and global) maximum. Maximization of ℓ can be attempted by gradient ascent, although there are more elaborate alternatives based on the Hessian matrix [108]. Gradient ascent iteratively updates an estimate $\mathbf{w}$ by the rule $\mathbf{w} \leftarrow \mathbf{w} + \eta \nabla_{\mathbf{w}} \ell(\mathbf{w})$, where η is the *learning rate*.

For any k, the partial derivative $\partial \ell(\mathbf{w}) / \partial w_k$ of ℓ at $\mathbf{w}$ with respect to the variable w_k is given by

$$\frac{\partial \ell(\mathbf{w})}{\partial w_k} = \sum_{i=1}^{N} y_i \frac{\partial}{\partial w_k} \left[\log \sigma(\mathbf{w}\mathbf{x}_i)\right] + (1 - y_i) \frac{\partial}{\partial w_k} \left[\log(1 - \sigma(\mathbf{w}\mathbf{x}_i))\right] \tag{2.14}$$

$$= \sum_{i=1}^{N} y_i \frac{\frac{\partial}{\partial w_k}\left[\sigma(\mathbf{w}\mathbf{x}_i)\right]}{\sigma(\mathbf{w}\mathbf{x}_i)} + (1 - y_i) \frac{\frac{\partial}{\partial w_k}\left[1 - \sigma(\mathbf{w}\mathbf{x}_i)\right]}{1 - \sigma(\mathbf{w}\mathbf{x}_i)} \tag{2.15}$$

$$= \sum_{i=1}^{N} y_i (1 - \sigma(\mathbf{w}\mathbf{x}_i)) x_{i,k} - (1 - y_i) \sigma(\mathbf{w}\mathbf{x}_i) x_{i,k} \tag{2.16}$$

$$= \sum_{i=1}^{N} (y_i - \sigma(\mathbf{w}\mathbf{x}_i)) x_{i,k}, \tag{2.17}$$

using the fact that $\sigma'(t) = \sigma(t)(1 - \sigma(t))$.

Thus, the gradient of ℓ with respect to $\mathbf{w}$ is given by

$$\nabla_{\mathbf{w}} \ell(\mathbf{w}) = \mathbf{X}^T (\mathbf{y} - \sigma(\mathbf{X}\mathbf{w})), \tag{2.18}$$

where the sigmoid function σ is applied element-wise, $\mathbf{X}$ is the design matrix (where each observation corresponds to a row), and $\mathbf{y} = (y_1, \ldots, y_N)$. This completes the description of logistic regression.

However, the formulation of logistic regression presented above can lead to extremely large coefficients in the resulting parameter estimate $\hat{\mathbf{w}}$, particularly when the data is linearly separable, which may cause overfitting. It is possible to penalize large coefficients by a hyperparameter λ, leading to the task of maximizing the regularized log-likelihood ℓ_λ given by

$$\ell_\lambda(\mathbf{w}) = \ell(\mathbf{w}) - \lambda ||\mathbf{w}||^2. \tag{2.19}$$

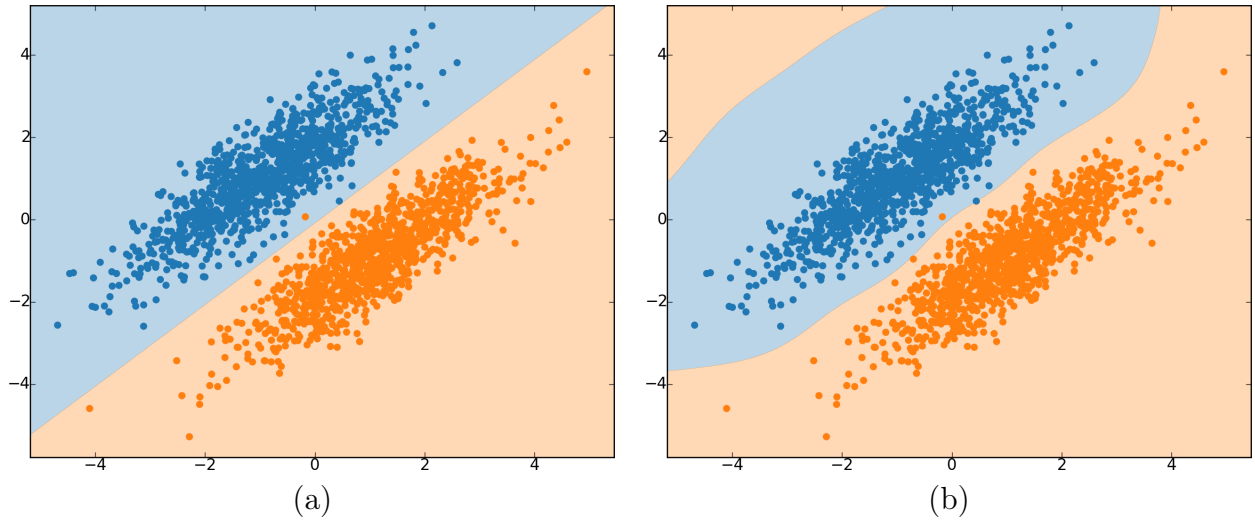


Figure 2.4: Soft-margin support vector machine ($C = 1$) decision boundary for a 2D dataset. (a) Linear kernel. (b) RBF kernel ($\gamma = 0.5$).

This objective function also has a (Bayesian) probabilistic interpretation (under a particular prior density function for $\mathbf{W}$), although we refer to [108] for details. If each observation was prefixed by the constant 1, the corresponding weight should not be penalized, since it does not make the classifier more sensitive to small changes in its inputs. Logistic regression can also be generalized to deal with multi-class problems [108].

In summary, logistic regression is appropriate for binary classification when there is a hyperplane that separates observations from distinct classes. In such cases, the data is considered *linearly separable*. Regularization is commonly employed to discourage large coefficients that may cause overfitting, which introduces a hyperparameter λ. Given an appropriate choice of feature map $\boldsymbol{\phi}$, logistic regression may also be applied when the data is not linearly separable.

In Section 2.4.2, we discuss how logistic regression may be applied to feature selection.

2.3.3 Support vector machines

Consider the iid dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, where $y_i \in \{-1, 1\}$, and the task of binary classification. A hard-margin support vector machine (SVM) is a classification technique that assumes that there is a hyperplane that separates the observations in such dataset by class [15]. Furthermore, the technique finds the separating hyperplane such that its distance to the nearest observation is maximum, a choice motivated by statistical learning theory [15]. For an illustrative example, see Fig. 2.4a.

Any (affine) hyperplane S in $\mathbb{R}^D$ can be written as $S = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{w}\mathbf{x} + b = 0\}$, for some nonzero weight vector $\mathbf{w} \in \mathbb{R}^D$, and intercept $b \in \mathbb{R}$. Consider any $\mathbf{x}, \mathbf{x}' \in S$. Clearly, $\mathbf{w}\mathbf{x} + b = \mathbf{w}\mathbf{x}' + b$, which implies $\mathbf{w}(\mathbf{x} - \mathbf{x}') = 0$. Intuitively, $\mathbf{w}$ is orthogonal to any vector pointing between vectors in S.

Consider any vector $\mathbf{x} \in \mathbb{R}^D$, and let $\mathbf{x}_\perp$ denote its closest vector in S. Recall that $\mathbf{x}$

can be written as [8]

$$\mathbf{x} = \mathbf{x}_\perp + r\frac{\mathbf{w}}{||\mathbf{w}||}, \tag{2.20}$$

where $r \in \mathbb{R}$ is the so-called signed distance between $\mathbf{x}$ and S. By introducing $\mathbf{w}$ and b into the equation above,

$$\mathbf{w}\mathbf{x} + b = \mathbf{w}\mathbf{x}_\perp + b + r\frac{||\mathbf{w}||^2}{||\mathbf{w}||} \tag{2.21}$$

$$\frac{\mathbf{w}\mathbf{x} + b}{||\mathbf{w}||} = r, \tag{2.22}$$

since $\mathbf{w}\mathbf{x}_\perp + b = 0$. Notice that the signed distance between S and the origin is given by $\frac{b}{||\mathbf{w}||}$.

Consider again the dataset $\mathcal{D}$, and the task of finding the parameters $\mathbf{w}$ and b of a separating hyperplane S such that $\mathbf{w}\mathbf{x}_i + b > 0$ if $y_i = 1$ and $\mathbf{w}\mathbf{x}_i + b < 0$ if $y_i = -1$, for all i, assuming that such hyperplane exists. This task is equivalent to finding parameters that satisfy the constraint $y_i(\mathbf{w}\mathbf{x}_i + b) > 0$, for all i.

The margin $m(\mathbf{w}, b)$ of a hyperplane $S = \{\mathbf{x} \in \mathbb{R}^D \mid \mathbf{w}\mathbf{x} + b = 0\}$ that satisfies the constraints is defined as

$$m(\mathbf{w}, b) = \min_i \frac{y_i(\mathbf{w}\mathbf{x}_i + b)}{||\mathbf{w}||} = \frac{1}{||\mathbf{w}||}\min_i y_i(\mathbf{w}\mathbf{x}_i + b). \tag{2.23}$$

Because we assumed that the hyperplane S satisfies the constraints, the margin corresponds to the (unsigned) distance between the hyperplane S and the closest observation in $\mathcal{D}$.

Notice that if the hyperplane S defined by $\mathbf{w}$ and b satisfies the constraints, then the hyperplane S' defined by $\kappa\mathbf{w}$ and κb also satisfies the constraints, for any $\kappa > 0$. Both S and S' also have the same margin, since

$$m(\kappa\mathbf{w}, \kappa b) = \min_i \frac{y_i(\kappa\mathbf{w}\mathbf{x}_i + \kappa b)}{||\kappa\mathbf{w}||} = \frac{\kappa}{\kappa||\mathbf{w}||}\min_i y_i(\mathbf{w}\mathbf{x}_i + b). \tag{2.24}$$

Consider the task of finding the parameters $\mathbf{w}^*$ and b^* of a separating hyperplane with maximum margin. In other words, maximizing $m(\mathbf{w}, b)$ with respect to $\mathbf{w}$ and b subject to $y_i(\mathbf{w}\mathbf{x}_i + b) > 0$, for all i. The margin $m(\mathbf{w}^*, b^*)$ of such a hyperplane is given by

$$m(\mathbf{w}^*, b^*) = \frac{1}{||\mathbf{w}^*||}\min_i y_i(\mathbf{w}^*\mathbf{x}_i + b^*) = \max_{\mathbf{w}}\max_b \frac{1}{||\mathbf{w}||}\min_i y_i(\mathbf{w}\mathbf{x}_i + b). \tag{2.25}$$

Consider also the task of maximizing $m(\mathbf{w}, b)$ with respect to $\mathbf{w}$ and b subject to $\min_i y_i(\mathbf{w}\mathbf{x}_i + b) = 1$. This constraint is clearly stronger than the constraint that $y_i(\mathbf{w}\mathbf{x}_i + b) > 0$, for all i. Suppose that a hyperplane S defined by $\mathbf{w}$ and b satisfies the weaker constraints, and let $\kappa = 1/\min_i y_i(\mathbf{w}\mathbf{x}_i + b)$, where the denominator is certainly positive. Consider the hyperplane S' defined by $\kappa\mathbf{w}$ and κb. As shown previously, $m(\mathbf{w}, b) =$

$m(\kappa\mathbf{w}, \kappa b)$. Since

$$\min_i \kappa y_i(\mathbf{w}\mathbf{x}_i + b) = \min_i \frac{y_i(\mathbf{w}\mathbf{x}_i + b)}{\min_j y_j(\mathbf{w}\mathbf{x}_j + b)} = 1, \tag{2.26}$$

any hyperplane S that satisfies the weaker constraints has a corresponding hyperplane S' with the same margin that satisfies the stronger constraint. Thus, we can maximize $m(\mathbf{w}, b)$ with respect to $\mathbf{w}$ and b subject to the stronger constraint without loss of generality.

The constraint $\min_i y_i(\mathbf{w}\mathbf{x}_i + b) = 1$ implies that the maximum margin hyperplane defined by $\mathbf{w}^*$ and b^* is given by

$$m(\mathbf{w}^*, b^*) = \max_{\mathbf{w}} \max_b \frac{1}{||\mathbf{w}||} \min_i y_i(\mathbf{w}\mathbf{x}_i + b) = \max_{\mathbf{w}} \frac{1}{||\mathbf{w}||}. \tag{2.27}$$

Instead, we consider the equivalent, and more convenient, task of minimizing $\frac{1}{2}||\mathbf{w}||^2$ with respect to $\mathbf{w}$ and b subject to the same constraint.

We will change constraints one last time. Consider minimizing $\frac{1}{2}||\mathbf{w}||^2$ with respect to $\mathbf{w}$ and b subject to $y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1$, for all i. These constraints are apparently weaker than the previous. However, suppose that $\mathbf{w}$ and b minimize $\frac{1}{2}||\mathbf{w}||^2$, and $y_i(\mathbf{w}\mathbf{x}_i + b) > 1$, for all i. Let $\kappa = 1/\min_i y_i(\mathbf{w}\mathbf{x}_i + b)$. A hyperplane defined by $\kappa\mathbf{w}$ and κb satisfies the new constraints, since $y_i(\kappa\mathbf{w}\mathbf{x}_i + \kappa b) = \kappa y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1$, for all i. However, $\frac{1}{2}||k\mathbf{w}||^2 < \frac{1}{2}||\mathbf{w}||^2$, since $0 < \kappa < 1$, which is a contradiction because we assumed $\mathbf{w}$ and b corresponded to a minimum. Therefore, a minimum that satisfies the new constraints also satisfies the previous constraint.

In summary, the parameters of the separating hyperplane with maximum margin are given by minimizing $\frac{1}{2}||\mathbf{w}||^2$ with respect to $\mathbf{w}$ and b subject to $y_i(\mathbf{w}\mathbf{x}_i + b) \geq 1$, for all i. Given the optimum $\mathbf{w}$ and b, a new observation $\mathbf{x}$ can be classified as 1 if $\mathbf{w}\mathbf{x} + b > 0$, and as -1 otherwise.

The optimization task stated above is a convex quadratic programming problem [15], a class of widely studied optimization problems. However, an important aspect of support vector machines requires restating this optimization task using the Lagrangian dual, which we cover next.

Let the generalized Lagrangian L corresponding to our problem be given by

$$L(\mathbf{w}, b, \mathbf{a}) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{N} a_i\big[y_i(\mathbf{w}\mathbf{x}_i + b) - 1\big]. \tag{2.28}$$

The Karush-Kuhn-Tucker conditions state that if $\mathbf{w}$ and b correspond to a local minimum subject to the constraints, then $\nabla L(\mathbf{w}, b, \mathbf{a}) = \mathbf{0}$ for some vector $\mathbf{a}$ whose elements are all

nonnegative [14]. The relevant gradients are given by

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b, \mathbf{a}) = \mathbf{w} - \sum_{i=1}^{N} a_i y_i \mathbf{x}_i \tag{2.29}$$

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \mathbf{a}) = \sum_{i=1}^{N} a_i y_i. \tag{2.30}$$

Therefore, if $\mathbf{w}$ and b correspond to a local minimum subject to the constraints, then

$$\mathbf{w} = \sum_{i=1}^{N} a_i y_i \mathbf{x}_i \tag{2.31}$$

$$0 = \sum_{i=1}^{N} a_i y_i, \tag{2.32}$$

for some $\mathbf{a}$ whose elements are all nonnegative.The Lagrangian dual $\tilde{L}$ is obtained by substituting the expressions above into the generalized Lagrangian L, and is given by

$$\tilde{L}(\mathbf{a}) = \sum_{i=1}^{N} a_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} a_i a_j y_i y_j \mathbf{x}_i \mathbf{x}_j. \tag{2.33}$$

It can be shown that if $\mathbf{a}$ maximizes $\tilde{L}$ subject to $a_i \geq 0$, for all i, and $\sum_i a_i y_i = 0$, then the corresponding $\mathbf{w} = \sum_i a_i y_i \mathbf{x}_i$ is the desired local minimum subject to constraints, and thus represents the maximum margin separating hyperplane [15].

The support vectors are the observations in $\mathcal{D}$ that are closest to the maximum margin separating hyperplane. It can be shown that only the coefficients a_i associated to these vectors are nonzero [15]. Therefore, given the expression for $\mathbf{w}$ in terms of $\mathbf{a}$, only the support vectors directly affect classification. Intuitively, the other observations in $\mathcal{D}$ can move freely as long as they do not affect the margin, resulting in the same maximum margin separating hyperplane. The intercept b can be obtained by noting that $y_i(\mathbf{w}\mathbf{x}_i + b) = 1$, for any support vector $\mathbf{x}_i$.

Although maximizing the Lagrangian dual is also a quadratic programming problem, it has a highly valuable characteristic. As with logistic regression, it would be possible to transform each input observation $\mathbf{x}$ by a pre-defined feature map $\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^{D'}$. In that case, the maximum margin separating hyperplane in the transformed space would not necessarily be a hyperplane in the original space.

However, notice that the observations in $\mathcal{D}$ only affect the Lagrangian dual through inner products. Let a kernel $k : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ be a function given by $k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})\boldsymbol{\phi}(\mathbf{x}')$, for a pre-defined feature map $\boldsymbol{\phi}$, and any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^D$. Clearly, as long as the kernel k is known, it is possible to find the desired maximum margin separating hyperplane in the transformed space without ever evaluating $\boldsymbol{\phi}$ directly. This is the so-called kernel trick, which vastly extends the applicability of support vector machines (and other techniques)

[15]. The same trick can be applied for classifying new observations, since

$$\mathbf{w}\boldsymbol{\phi}(\mathbf{x}) + b = \sum_i a_i y_i \boldsymbol{\phi}(\mathbf{x}_i)\boldsymbol{\phi}(\mathbf{x}) + b = \sum_i a_i y_i k(\mathbf{x}_i, \mathbf{x}) + b, \tag{2.34}$$

for any $\mathbf{x} \in \mathbb{R}^D$.

There are many known sufficient conditions for defining valid kernels disregarding $\boldsymbol{\phi}$ [15]. A common choice is the radial basis function kernel k given by

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma||\mathbf{x} - \mathbf{x}'||^2), \tag{2.35}$$

where γ is a hyperparameter. For an illustrative example, see Fig. 2.4b. A support vector machine whose kernel corresponds to the inner product in the original feature space is said to be linear.

There are two very simple strategies to adapt support vector machines for classification problems with $C > 2$ classes. One of them is to train C one-vs-rest classifiers, and to classify a new observation by the class for which the observation is further away from the corresponding maximum margin separating hyperplane (on the correct side). Another heuristic is to train $C(C-1)/2$ one-vs-one classifiers, and to classify a new observation according to the class that receives more *votes*. Both strategies offer few guarantees in the general case [15].

A soft-margin support vector machine is a technique that finds a separating hyperplane that may leave some observations on the *wrong side*, as long as doing so increases the margin. The trade-off is controlled by a penalty hyperparameter (typically denoted by C). Intuitively, this alternative formulation is more robust to outliers [15].

In summary, in its simplest formulation, a support vector machine is appropriate for binary classification of linearly separable data. The introduction of feature maps through the kernel trick easily allows binary classification of data that is not linearly separable, and requires choosing a kernel function and its hyperparameters. Soft-margins make support vector machines generally more robust to outliers, introducing yet another hyperparameter C. Finally, support vector machines can be adapted to multi-class classification tasks through simple heuristics.

In Section 2.4.3, we discuss how support vector machines can be applied to feature selection.

2.3.4 Decision trees

A (classification) decision tree assigns observations to classes according to a sequence of logical tests that involve their features. We focus on building classification trees that perform inequality tests using the so-called CART (classification and regression trees) approach [64].

A classification tree is a full rooted binary tree $T = (V, E)$. By definition, T either has a single vertex (root), or can be built by connecting (by two edges) a single vertex (root) to the roots of two other binary trees.

Consider the task of classifying observations in $\mathbb{R}^D$. Each vertex u in the classification

tree T is associated to a set $R_u \subseteq \mathbb{R}^D$. Suppose v and w are children of u. The vertex v is associated to the set $R_v = \{\mathbf{x} \in R_u \mid x_j < \tau\}$, and w is associated to the set $R_w = \{\mathbf{x} \in R_u \mid x_j \geq \tau\}$, for some feature j and constant threshold τ. The root r is associated to the set $R_r = \mathbb{R}^D$. Therefore, the leaves of T partition $\mathbb{R}^D$ into hyperrectangular regions. If each leaf of T is also associated to a class, an observation $\mathbf{x}$ can be classified by finding the leaf u such that $\mathbf{x} \in R_u$, by following the appropriate branches starting from the root. For an illustrative example, see Fig. 2.5.

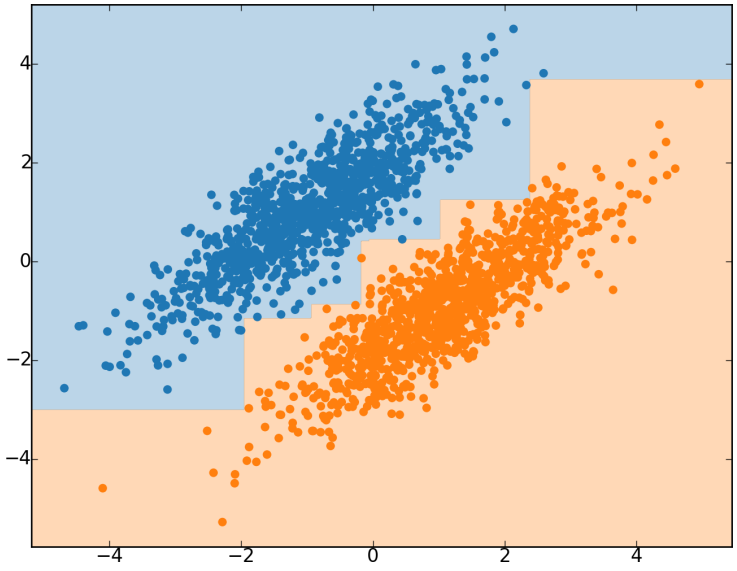


Figure 2.5: Decision tree decision boundary for a 2D dataset.

Consider the iid dataset $\mathcal{D}$, and the task of building a classification tree that is expected to generalize well to new observations. A typical strategy is to find a pure classification tree T, whose leaves are all pure. A leaf u is pure if R_u contains only observations from $\mathcal{D}$ that belong to the same class [108].

Suppose a dataset has as many distinct classes as there are observations. In that case, define the cost of a pure classification tree T for the dataset $\mathcal{D}$ as the cumulative number of tests required to classify each observation in $\mathcal{D}$ using T. Determining whether there is a classification tree with cost less or equal to k given $\mathcal{D}$ and pre-defined pairs of features and thresholds is an NP-complete problem [72], among other related problems. The computational burden of finding *optimal* trees is typically avoided by employing greedy heuristic methods.

We now describe a common heuristic to choose the feature j^* and threshold τ^* for the root of a classification tree given the dataset $\mathcal{D}$.

For each feature j and threshold τ, define the cost $C(j, \tau)$ of partitioning a dataset $\mathcal{D}$ into non-empty datasets $\mathcal{D}_v = \{(\mathbf{x}, y) \in \mathcal{D} \mid x_j < \tau\}$ and $\mathcal{D}_w = \mathcal{D} - \mathcal{D}_v = \{(\mathbf{x}, y) \in \mathcal{D} \mid x_j \geq \tau\}$ as

$$C(j, \tau) = c(\mathcal{D}_v) + c(\mathcal{D}_w), \tag{2.36}$$

where c is a pre-defined cost function, which we discuss later. We choose by exhaustive

search a feature j^* and threshold τ^* such that

$$C(j^*, \tau^*) = \min_j \min_\tau C(j, \tau). \tag{2.37}$$

Notice that the cost C of at most $D(N-1)$ pairs of features and thresholds needs to be computed to find j^* and τ^*, even if the features are real-valued (since at most $N-1$ distinct thresholds would affect the partitioning of $\mathcal{D}$ into non-empty $\mathcal{D}_v$ and $\mathcal{D}_w$, for each feature).

A typical choice of cost function c is the entropy of the empirical distribution of classes in $\mathcal{D}$, given by [108, 64]

$$c(\mathcal{D}) = -\sum_y \pi_y \log \pi_y, \tag{2.38}$$

where π_y is the fraction of observations in $\mathcal{D}$ that belongs to class y, and $0 \log 0$ is substituted by 0. Intuitively, such entropy is minimized whenever all observations in $\mathcal{D}$ belong to the same class. We discuss entropy in more detail in Section 2.4.1.

After choosing j^* and τ^* for the root vertex, its children u and w can be seen as roots of two distinct classification trees. By restricting each children v to its corresponding dataset $\mathcal{D}_v \subset R_v$, the procedure described above can be applied recursively. The procedure should not create children for a root when the dataset is already pure, since that would not affect future classifications. This completes a recursive algorithm for building a classification tree.

It is also possible to stop creating children whenever a pre-defined tree depth is achieved, or whenever the remaining dataset contains few observations [64]. In such cases, a new observation $\mathbf{x}$ is classified according to the class majority in the leaf u such that $\mathbf{x} \in R_u$. The objective of these stop criteria is to make the classifier more robust to small changes in the training data. Another way to prevent overfitting is to *prune* the resulting classification tree, by eliminating branches according to their effect on training set accuracy [64]. In all of these cases, the associated hyperparameters may be chosen by cross-validation.

Classification trees have some highly desirable properties. For instance, they are insensitive to monotonic transformation of features, since they are based on thresholds. More importantly, classification trees are more interpretable than many other classifiers, whose outputs cannot be easily understood in terms of the original features [64].

Consider sampling N elements (with replacement) from the elements in the dataset $\mathcal{D}$ to create each dataset in a sequence $\mathcal{D}_1, \ldots, \mathcal{D}_S$, and training a distinct classification tree for each dataset in this sequence. Suppose also that any observation $\mathbf{x} \in \mathbb{R}^D$ is classified according to the class that receives more votes from the S independent classifiers. This strategy is called bagging, and the meta-classifier is a type of ensemble. Intuitively, aggregating votes from classification trees trained using distinct datasets is typically more robust than depending on a classification tree that learned rules that may be overly specific for a particular dataset [23].

A random forest classifier (RFC) is an ensemble of classification trees based on bagging [23]. Each classification tree in the ensemble also considers only a random subset of $d < D$ features in each step of finding a (locally) optimum pair of feature j^* and threshold τ^*.

As before, the goal is to create robust meta-classifiers.

Similarly to a random forest classifier, an extremely randomized tree classifier is an ensemble classifier that introduces randomness into the learning process in an attempt to reduce overfitting [51]. However, extremely randomized trees typically do not perform bagging. Instead, the technique fits S distinct classification trees to the entire dataset. In each step that requires choosing a feature j^* and a threshold τ^* to *split* the remaining dataset $\mathcal{D}$, only a random subset $\mathcal{F} \subseteq \{1, \ldots, D\}$ containing $d \leq D$ features is considered. Furthermore, for each feature $j \in \mathcal{F}$, a single threshold τ is chosen uniformly at random in the range of j (in the remaining dataset $\mathcal{D}$). From these candidates, the feature j^* and threshold τ^* with the lowest cost $C(j^*, \tau^*)$ are chosen, as usual.

Both random forests and extremely randomized trees have shown remarkable empirical efficacy results [23, 51].

In summary, although building pure classification trees requires few choices, such as the cost function c, it is generally important to introduce hyperparameters that control tree *complexity*. Ensembles of classification trees are particularly effective, and introduce even more choices.

In Section 2.4.4, we discuss how extremely randomized trees can be applied to feature selection.

2.3.5 Artificial neural networks

Advances in computational power and techniques for building and training artificial neural networks have allowed these models to achieve state-of-the-art results in many applications related to pattern recognition [130]. We will present two models of feedforward neural networks: multilayer perceptrons (due to their simplicity), and convolutional neural networks (due to their state-of-the-art image classification results). These models are the focus of Chapter 5.

Multilayer perceptrons

Multilayer perceptrons compose the most widely known class of artificial neural networks [15, 112].

Consider an iid dataset $\mathcal{D} = (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$, where $\mathbf{x}_i \in \mathbb{R}^D$, and $\mathbf{y}_i \in \{0, 1\}^C$. In this context, given a pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, we assume $y_j = 1$ if and only if observation $\mathbf{x}$ belongs to class j. As usual, we also assume that each observation belongs to a single class.

Let L represent the number of layers in the network, and $N^{(l)}$ represent the number of neurons in layer l, with $N^{(L)} = C$. These hyperparameters determine the so-called network architecture. We will refer to a neuron in layer l by a corresponding number between 1 and $N^{(l)}$. The neurons in the first layer are also called input units, the neurons in the output (last) layer called output units, and the other neurons called hidden units. Networks with more than 3 layers are called deep networks.

Let $w_{j,k}^{(l)} \in \mathbb{R}$ represent the weight reaching neuron j in layer l from neuron k in layer $(l-1)$. The order of the indices is counterintuitive, but makes the presentation simpler. Furthermore, let $b_j^{(l)} \in \mathbb{R}$ represent the bias for neuron j in layer l.

Consider a layer l, for $1 < l \leq L$, and neuron j, for $1 \leq j \leq N^{(l)}$. The weighted input to neuron j in layer l is defined as

$$z_j^{(l)} = b_j^{(l)} + \sum_{k=1}^{N^{(l-1)}} w_{j,k}^{(l)} a_k^{(l-1)}, \tag{2.39}$$

where the activation $a_j^{(l)}$ of neuron j in layer $1 < l < L$ is defined as

$$a_j^{(l)} = \sigma(z_j^{(l)}), \tag{2.40}$$

for some differentiable activation function σ. We consider the sigmoid activation function defined by $\sigma(z) = \frac{1}{1+e^{-z}}$.

We will also consider a so-called softmax output layer, where the activation $a_j^{(L)}$ of neuron j is given by

$$a_j^{(L)} = \frac{e^{z_j^{(L)}}}{\sum_{k=1}^{C} e^{z_k^{(L)}}}. \tag{2.41}$$

It will be useful to define some vectors and matrices that represent quantities associated to each neuron in a given layer. The weighted input for layer $l > 1$ is defined as $\mathbf{z}^{(l)} = (z_1^{(l)}, \ldots, z_{N^{(l)}}^{(l)})$, and the activation vector for layer l is defined as $\mathbf{a}^{(l)} = (a_1^{(l)}, \ldots, a_{N^{(l)}}^{(l)})$. Furthermore, we define the bias vectors as $\mathbf{b}^{(l)} = (b_1^{(l)}, \ldots, b_{N^{(l)}}^{(l)})$, and the $N^{(l)} \times N^{(l-1)}$ weight matrices $\mathbf{W}^{(l)}$ such that $\mathbf{W}_{j,k}^{(l)} = w_{j,k}^{(l)}$.

Using these definitions, the output of each layer $1 < l < L$ can be written as

$$\mathbf{a}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}), \tag{2.42}$$

where the activation function is applied element-wise (see Fig. 2.6).

The output of a multilayer perceptron when $\mathbf{a}^{(1)} = \mathbf{x}$ is defined as the activation vector $\mathbf{a}^{(L)}$ of the output layer. Notice how $\mathbf{a}^{(L)}$ is implicitly dependent on $\mathbf{x}$.

This completes the definition of the model. We now focus on learning parameters for classification given a dataset.

Suppose the dataset $\mathcal{D}$ is iid according to $p(\cdot \mid \boldsymbol{\theta}^*)$, for an unknown parameter vector $\boldsymbol{\theta}^*$. Furthermore, suppose the probability of any class y given any observation $\mathbf{x}$ is given by the corresponding output neuron of a particular multilayer perceptron with a fixed architecture when $\mathbf{a}^{(1)} = \mathbf{x}$. More concretely, suppose

$$p(y \mid \mathbf{x}, \boldsymbol{\theta}^*) = a_y^{(L)}, \tag{2.43}$$

such that the unknown $\boldsymbol{\theta}^*$ defines the parameters (weights and biases) of a multilayer perceptron (among the prior probabilities of observations, which are irrelevant for our purposes). It is important to notice that a softmax output layer would yield a valid probability mass function for any choice of observation, weights and biases.

The (conditional) likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$ of the parameter vector $\boldsymbol{\theta}$ given the dataset $\mathcal{D}$

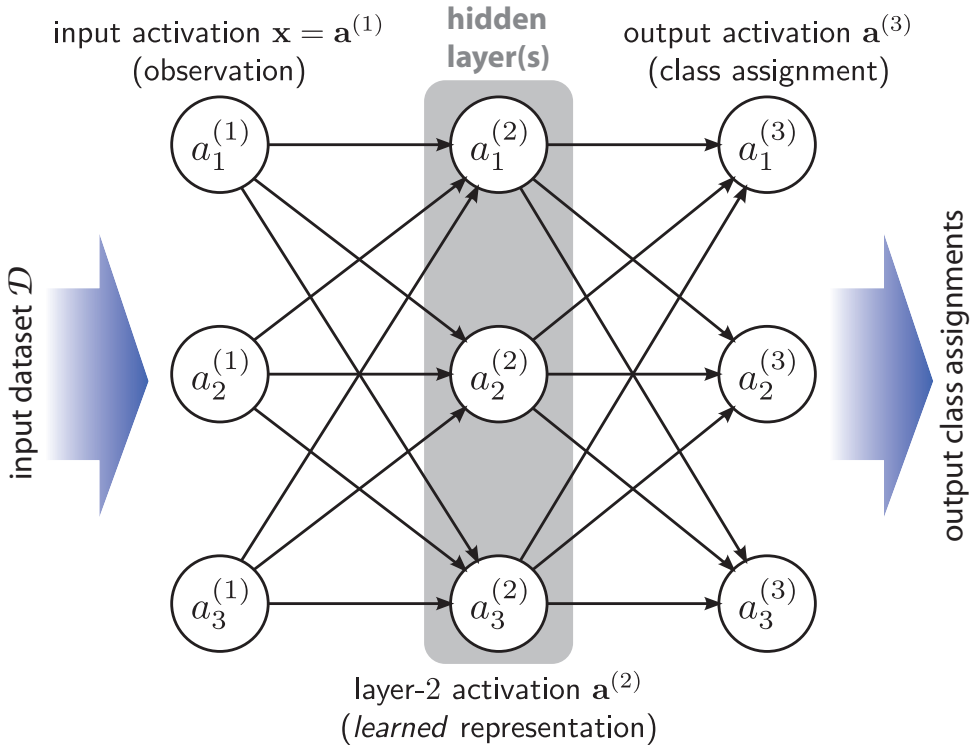


Figure 2.6: Schema of multilayer perceptron with three layers and three neurons per layer.

may be written as

$$p(\mathcal{D} \mid \boldsymbol{\theta}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \boldsymbol{\theta}), \tag{2.44}$$

by ignoring for a moment that we encoded the target classes using vectors. Once again, this likelihood is not based on the joint density over observations and classes, which is irrelevant for our purposes.

A natural goal is to find the weights and biases that minimize the (average) negative log-likelihood J, which is given by

$$J = -\frac{1}{N} \log p(\mathcal{D} \mid \boldsymbol{\theta}) = -\frac{1}{N} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \sum_{k=1}^{C} y_k \log a_k^{(L)}, \tag{2.45}$$

where $\mathbf{a}^{(L)}$ is the output activation when the network parameterized according to $\boldsymbol{\theta}$ receives $\mathbf{x}$ as input. Notice that a single y_k is nonzero inside the second summation. Furthermore, notice that $-y_k \ln a_k^{(L)} \to \infty$ when $y_k = 1$ and $a_k^{(L)} \to 0$, which characterizes a prediction error ($a_k^{(L)} > 0$ due to the softmax output layer).

If we let $E = -\sum_k y_k \log a_k^{(L)}$ be a cost variable implicitly associated to a pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, then $J = N^{-1} \sum_{(\mathbf{x},\mathbf{y})} E$. The fact that the cost J can be written as an average of costs E for each element of the dataset will be crucial to the proposed optimization procedure. The procedure requires the computation of partial derivatives of the cost with respect to weights and biases, which are usually computed by a technique called backpropagation .

Let the error[1] of neuron j in layer l for a given $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ be defined as

$$\delta_j^{(l)} = \frac{\partial E}{\partial z_j^{(l)}}. \tag{2.46}$$

Similarly, the error for the neurons in layer l is denoted by $\boldsymbol{\delta}^{(l)} = \left(\delta_1^{(l)}, \ldots, \delta_{N^{(l)}}^{(l)}\right)$.

Backpropagation is a method for computing the partial derivatives of the cost function of a multilayer perceptron with respect to its parameters [112]. Given our choice of activation functions, the method is based solely on the following six equalities:

$$\boldsymbol{\delta}^{(L)} = \mathbf{a}^{(L)} - \mathbf{y}, \tag{2.47}$$

$$\boldsymbol{\delta}^{(l)} = ((\mathbf{W}^{(l+1)})^T \boldsymbol{\delta}^{(l+1)}) \odot \sigma'(\mathbf{z}^{(l)}), \tag{2.48}$$

$$\frac{\partial E}{\partial b_j^{(l)}} = \delta_j^{(l)}, \tag{2.49}$$

$$\frac{\partial E}{\partial w_{j,k}^{(l)}} = a_k^{(l-1)} \delta_j^{(l)}, \tag{2.50}$$

$$\frac{\partial J}{\partial b_j^{(l)}} = \frac{1}{N} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \frac{\partial E}{\partial b_j^{(l)}}, \tag{2.51}$$

$$\frac{\partial J}{\partial w_{j,k}^{(l)}} = \frac{1}{N} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \frac{\partial E}{\partial w_{j,k}^{(l)}}, \tag{2.52}$$

where $\odot$ denotes element-wise multiplication. Notice how every quantity on the right side can be computed easily from our definitions, by starting with the errors in the output layer for every pair $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$. This originates the term backpropagation.

As an illustration, we will demonstrate Eq. 2.48, which states that

$$\delta_j^{(l)} = \sigma'(z_j^{(l)}) \sum_{k=1}^{N^{(l+1)}} w_{k,j}^{(l+1)} \delta_k^{(l+1)}, \tag{2.53}$$

for $1 < l < L$ and $1 \leq j \leq N^{(l)}$. Since layer $l < L$ only affects the output through the next layer, and $z_k^{(l+1)}$ is a differentiable function of $z_1^{(l)}, \ldots, z_{N^{(l)}}^{(l)}$, and E is a differentiable function of $z_1^{(l+1)}, \ldots, z_{N^{(l+1)}}^{(l+1)}$,

$$\delta_j^{(l)} = \frac{\partial E}{\partial z_j^{(l)}} = \sum_{k=1}^{N^{(l+1)}} \frac{\partial E}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{N^{(l+1)}} \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}}. \tag{2.54}$$

By definition,

$$z_k^{(l+1)} = b_k^{(l+1)} + \sum_{i=1}^{N^{(l)}} w_{k,i}^{(l+1)} a_i^{(l)}, \tag{2.55}$$

[1] This arguably misleading term is widely employed [112].

therefore,

$$\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \frac{\partial}{\partial z_j^{(l)}}\left[w_{k,j}^{(l+1)} a_j^{(l)}\right] = w_{k,j}^{(l+1)} \sigma'(z_j^{(l)}). \tag{2.56}$$

This gives

$$\delta_j^{(l)} = \sum_{k=1}^{N^{(l+1)}} \delta_k^{(l+1)} w_{k,j}^{(l+1)} \sigma'(z_j^{(l)}) = \sigma'(z_j^{(l)}) \sum_{k=1}^{N^{(l+1)}} w_{k,j}^{(l+1)} \delta_k^{(l+1)}, \tag{2.57}$$

as we wanted to show.

Intuitively, for each observation, backpropagation considers the effect of a small increase of Δw on a parameter w in the network. This change affects every subsequent neuron on a path to the output, and ultimately changes the cost J by a small ΔJ.

Gradient descent can be used as a heuristic to find the parameters that minimize the cost J. More concretely, if we let $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ denote the gradient (direction of maximum local increase) of J given the parameter vector $\boldsymbol{\theta}$ (which represents weights and biases), the technique starts at a parameter vector $\boldsymbol{\theta}_0$ chosen arbitrarily, and visits the sequence of parameter vectors given by

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t), \tag{2.58}$$

where the learning rate $\eta \in \mathbb{R}^+$ is a small constant. Gradient descent is not guaranteed to converge. Even if it converges, the point at convergence may be a saddle point or a poor local minima. The choice of η considerably affects the success of gradient descent.

In a given iteration t of gradient descent, instead of computing $\frac{\partial J}{\partial w_{j,k}^{(l)}}$ and $\frac{\partial J}{\partial b_j^{(l)}}$ as averages derived from a computation involving all $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$, it is also possible to consider only a subset $\mathcal{D}' \subseteq \mathcal{D}$ of randomly chosen observations. The dataset $\mathcal{D}$ may also be partitioned randomly into subsets called batches, which are considered in sequence. In this case, another random partition is considered once every subset is used. This procedure, called mini-batches stochastic gradient descent, is widely used due to its efficiency [112]. Intuitively, the procedure makes faster decisions based on sampling. Regardless of these choices, a sequence of iterations that considers all observations in the dataset is called an epoch.

The basic choices involved in learning the parameters for a multilayer perceptron using mini-batches include at least the number of hidden layers, the number of neurons in each hidden layer, size of the mini-batches, the number of epochs, and the learning rate η.

The momentum technique is a common heuristic for training deep artificial neural networks [112]. In momentum-based stochastic gradient descent, each parameter w in the network (weight or bias) has a corresponding velocity v. The velocity is defined by $v_0 = 0$ and

$$v_{t+1} = \mu v_t - \eta \left[\frac{\partial J}{\partial w_t}\right], \tag{2.59}$$

where v_t and w_t correspond, respectively, to v and w at iteration t of stochastic gradient descent. At each iteration, the parameter w is updated by the rule $w_{t+1} = w_t + v_{t+1}$. Intuitively, the momentum technique remembers the velocity of each parameter, allowing

larger updates when the direction of decrease in cost is consistent over many iterations. The parameter $0 \leq \mu \leq 1$ controls the effect of the previous velocity on the next velocity, and $1 - \mu$ is commonly interpreted as a coefficient of friction. If $\mu = 0$, the technique is equivalent to gradient descent.

Dropout [137] is another common heuristic for training deep artificial neural networks. At every iteration of stochastic gradient descent, *half* the hidden neurons are removed at random. In most implementations, this can be accomplished by forcing the outputs of the corresponding neurons to be zero. The modified network is applied as usual to the observations in a mini-batch, and backpropagation follows, as if the network were not changed. The resulting partial derivatives are used to update the parameters of the neurons that were not removed. After training is finished, the weights incoming from hidden neurons are *halved*. This heuristic is believed to make the network robust to the absence of particular features, which might be particular to the training data [137]. Dropout is considered related to regularization for trying to reduce overfitting [137].

There are many more heuristics for implementing multilayer perceptrons that will not be described in detail in this text [112]. Although we focused most of the discussion on sigmoid neurons, rectified linear neurons have achieved superior results in important benchmarks [52].

In summary, training a multilayer perceptron involves a large number of hyperparameters, such as number of layers, number of neurons per layer, learning rate, momentum coefficient, mini-batch size, and number of epochs. The success of multilayer perceptrons is highly dependent on these hyperparameters, and their choice requires significant expertise [12]. We return to this issue in Chapter 5.

Convolutional neural networks

Convolutional neural networks were first developed for image classification, which is the focus of this section, although they have also been successfully applied to other tasks [130, 112].

A two-dimensional image may be represented by a function $f : \mathbb{Z}^2 \to \mathbb{R}^c$. An element $\mathbf{a} \in \mathbb{Z}^2$ is called a pixel, and $f(\mathbf{a})$ is the value of pixel $\mathbf{a}$. If $f(\mathbf{a}) = (f_1(\mathbf{a}), \ldots, f_c(\mathbf{a}))$, then f_i is called channel i.

A window $W \subset \mathbb{Z}^2$ is a finite set $W = [s_1, S_1] \times [s_2, S_2]$ that corresponds to a rectangle in the image domain. The size of this window W is denoted by $w \times h$, where $w = S_1 - s_1 + 1$ and $h = S_2 - s_2 + 1$. Because the domain Z of images of interest is usually a window, it is possible to *flatten* an image f into a vector $\mathbf{x} \in \mathbb{R}^{c|Z|}$. In this vector, there is a scalar value $f_i(\mathbf{a})$ corresponding to the value of each channel i of each pixel $\mathbf{a} \in Z$.

Consider an iid dataset $\mathcal{D} = (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$, such that $\mathbf{x}_i \in \mathbb{R}^D$, and $\mathbf{y}_i \in \{0, 1\}^C$, where each vector $\mathbf{x}_i$ corresponds to a distinct image $\mathbb{Z}^2 \mapsto \mathbb{R}^c$. Also, suppose that all images are defined on the same window Z, such that $D = c|Z|$. The task of image classification consists on assigning a class label for a given image based on generalization from $\mathcal{D}$.

A convolutional neural network is particularly well suited for image classification, because it explores the spatial relationships between pixels (organization in $\mathbb{Z}^2$) [112].

Similarly to multilayer perceptrons, a convolutional neural network is also a parameterized function, and the parameters are usually learned by stochastic gradient descent on a cost function defined on the training set. In contrast to multilayer perceptrons, there are three main types of layers in a convolutional neural network: convolutional layers, pooling layers and fully connected layers (see Fig. 2.7) [112].

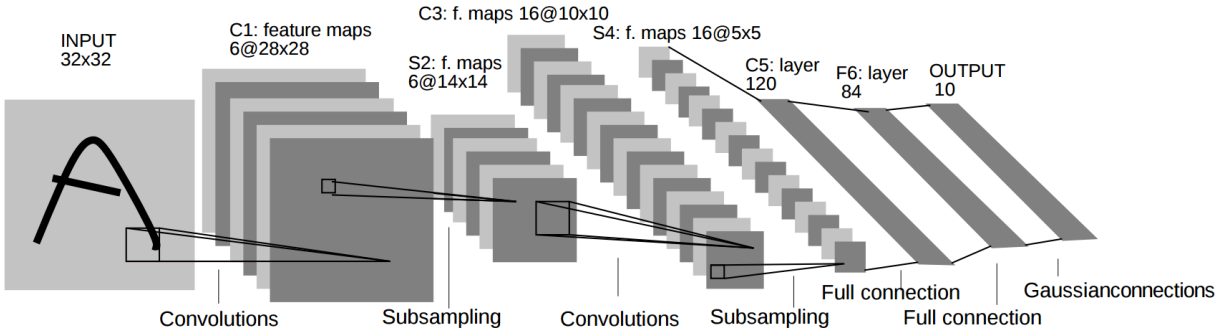


Figure 2.7: Schema of convolutional neural network including convolutional layers, pooling (sub-sampling) layers, and fully connected layers. Source: [85].

A convolutional layer receives an input image f and outputs an image o. A convolutional layer is composed solely of artificial neurons. Each artificial neuron h in a convolutional layer l receives as input the values in a window $W = [s_1, S_1] \times [s_2, S_2] \subset Z$ of size $w \times h$, where Z is the domain of f. The weighted output $z_h^{(l)}$ of that neuron is given by

$$z_h^{(l)} = b_h^{(l)} + \sum_{i=1}^{c} \sum_{j=s_1}^{S_1} \sum_{k=s_2}^{S_2} w_{h,i,j,k}^{(l)} a_{i,j,k}^{(l-1)}. \tag{2.60}$$

In the equation above, $a_{i,j,k}^{(l-1)} = f_i(j,k)$ is the value of pixel (j,k) in channel i of the input image. Also, $b_h^{(l)}$ is the bias of neuron h and $w_{h,i,j,k}^{(l)}$ is the weight that neuron h in layer l associates to $f_i(j,k)$. The activation function for a convolutional layer is typically rectified linear [112], so $a_h^{(l)} = \max(0, z_h^{(l)})$. The definition of $z_h^{(l)}$ is similar to the definition of the weighted input for a neuron in a multilayer perceptron. The only difference is that a neuron in a convolutional layer is not necessarily connected to the activations of all neurons in the previous layer, but only to the activations in a particular $w \times h$ window W. Each neuron in a convolutional layer has cwh weights and a single bias.

A neuron in a convolutional layer is replicated (through parameter *sharing*) for all windows of size $w \times h$ in the domain Z whose centers are offset by pre-defined steps. These steps are the horizontal and vertical *strides*. The activations corresponding to a neuron replicated in this way correspond to the values in a single channel of the output image o (appropriately arranged in $\mathbb{Z}^2$). Thus, an output image $o : \mathbb{Z}^2 \to \mathbb{R}^n$ is obtained by replicating n neurons over the whole domain of the input image. The total number of free parameters in a convolutional layer is only $n(cwh + 1)$. If the parameters in a convolutional layer were not shared by replicated neurons, the number of parameters would be $mn(cwh + 1)$, where m is the number of windows of size $w \times h$ that fit into f (for the given strides).

The weighted outputs (minus the bias) of replicated neurons correspond to an output channel that is analogous to the discrete (multichannel) convolution of the input f with a particular image g. The values of g correspond to the (shared) weights of the replicated neurons (appropriately arranged in $\mathbb{Z}^2$). This assumes that the horizontal and vertical strides are 1 and that the domain of the resulting image is always restricted to the window domain of f. In other words, each channel o_u in the output o of a convolutional layer corresponds to a (multichannel) convolution with an image g_u, which is also called a filter. This is the origin of the name convolutional network [112]. Therefore, to define a convolutional layer, it is enough to specify the size of the filters (window size), the number of filters (number of channels in the output image), horizontal and vertical strides (which are usually 1).

Each channel in the output of a convolutional layer can also be seen as the response of the input image to a particular (learned) filter. Based on this interpretation, each channel in the output image is also called an activation map.

Backpropagation can be adapted to compute the partial derivative of the cost with respect to every parameter in a convolutional layer [130, 112]. The fact that a single weight affects the output of several neurons must be taken into account. We omit the details of backpropagation for convolutional neural networks in this text.

A pooling layer receives an input image $f : \mathbb{Z}^2 \to \mathbb{R}^c$ and outputs an image $o : \mathbb{Z}^2 \to \mathbb{R}^c$. A pooling layer reduces the size of the window domain Z of f by an operation that acts independently on each channel. A typical pooling technique is called max-pooling [112]. In max-pooling, the maximum value of channel f_i in a particular window of size $w \times h$ corresponds to an output value in channel o_i. To define a max-pooling layer, it is enough to specify the size of these windows and the strides (which usually match the window dimensions). The objective of reducing the spatial domain of the image is to achieve similar results to using comparatively larger convolutional filters in the next layers [112]. This supposedly allows the detection of higher-level features in the input image with a reduced number of parameters [112]. It is also believed that max-pooling improves the invariance of the classification to translations of the original image [112]. In practice, a sequence of alternating convolutional and max-pooling layers has obtained excellent results in many image classification tasks [130, 135]. Backpropagation can also be performed through max-pooling layers.

In summary, a max-pooling layer receives an input image $f : \mathbb{Z}^2 \to \mathbb{R}^c$ and outputs an image $o : \mathbb{Z}^2 \to \mathbb{R}^c$ defined by

$$o_i(j,k) = \max_{\mathbf{a} \in W_{j,k}} f_i(\mathbf{a}), \tag{2.61}$$

where $i \in \{1, \ldots, c\}$, $(j,k) \in \mathbb{Z}^2$, Z is the window domain of f, and $W_{j,k} \subseteq Z$ is the input window corresponding to output pixel (j,k).

A fully connected layer receives an input image $f : \mathbb{Z}^2 \to \mathbb{R}^c$ or an input vector $\mathbf{x}$ and outputs a vector $\mathbf{o}$. A fully connected layer is precisely analogous to a layer in a multilayer perceptron [112], and can only be succeeded by other fully connected layers. The final layer in a convolutional neural network is always a fully connected layer with C neurons, which is responsible for representing the classification. Backpropagation in fully

connected layers is analogous to backpropagation in multilayer perceptrons.

Deep convolutional neural networks are usually trained in large labeled datasets, requiring (non-trivial) efficient implementations, which include all the improvements mentioned in the previous section [130]. After training a convolutional neural network for a particular dataset, it is possible to re-use the parameters of the network (up to its last fully connected layer) as a starting point for another classification task. This technique decouples *representation learning* from a specific image classification problem, and has been very successful in practice [130].

In summary, convolutional neural networks are highly specialized models that were originally conceived for image classification. The choice of hyperparameters (including types of layers, number of layers, filters per layer, filter sizes, strides, and the usual training procedure hyperparameters) is crucial for efficacy [130, 112, 135]. We return to this issue in Chapter 5.

2.4 Feature selection

As already mentioned, extracting appropriate features from the objects of interest is crucial to the success of pattern classification. Using too few features can lead to poor generalization, while using too many features can be computationally prohibitive, or even introduce confounding information into the training data [60, 93].

Because there are $2^D - 1$ distinct non-empty subsets of a set of D features, evaluating the efficacy of every feature subset using cross-validation is generally infeasible. Instead, feature selection is typically performed by heuristic methods. Feature selection techniques are typically divided into *wrappers*, which are based on learning algorithms, and *filters*, which rely on simpler metrics derived from the relationships between features and class labels. We refer to [93] for an extensive list of techniques.

The next sections introduce feature selection techniques (wrappers) that will be employed in Chapters 4 and 5. These methods were chosen for being sufficiently inexpensive (computationally) to aid in interactive feature selection.

One exception is Section 2.4.1, which introduces the concept of mutual information. Although mutual information is ideal to select (discrete) features individually, it fails to take into account whether features are important when considered together with others (unless feature subsets are considered, which becomes intractable). However, Sec. 2.4.1 is also used to introduce important concepts in information theory, which will be useful again in Sec. 2.6.4.

2.4.1 Mutual information

The entropy (in bits) $H[X]$ of a discrete random variable $X \sim p$ is given by [98]

$$H[X] = \mathbb{E}[-\log_2 p(X)] = -\sum_k p(k) \log_2 p(k), \tag{2.62}$$

where $p(k) > 0$ for every k. If $p(k) = 0$ for some k, $0 \log_2 0$ is conventionally replaced

by 0, which is justified by the fact that $\lim_{a\to 0} a \log_2 a = 0$. It is also common to denote $H[X]$ by $H[p]$.

Consider the task of transmitting an iid dataset $\mathcal{D} = x_1, \ldots, x_N$ distributed according to a probability mass function p. Shannon's source coding theorem states that, as $N \to \infty$, the entropy $H[p]$ is a lower bound on the average number of bits required to transmit each observation (after the encoding is established) [98].

The cross-entropy $H[p, q]$ between two probability mass functions p and q is defined as $H[p, q] = -\sum_k p(k) \log_2 q(k)$, where $0 \log_2 0$ is again replaced by 0. Considering the transmission task mentioned above, the cross-entropy $H[p, q]$ can be interpreted as the average number of bits required to transmit an observation when the encoding is ideal for q (instead of p) [108, 98].

The Kullback-Leibler divergence $\mathrm{KL}(p||q)$ between two probability mass functions p and q is defined as

$$\mathrm{KL}(p||q) = \sum_k p(k) \log_2 \frac{p(k)}{q(k)} = H[p, q] - H[p], \tag{2.63}$$

where $p(k), q(k) > 0$ for every k. If either $p(k)$ or $q(k)$ are 0, the corresponding term in the summation is again replaced by 0. Considering again the transmission task, the Kullback-Leibler divergence $\mathrm{KL}(p||q)$ can be interpreted as the increase in the average number of bits required to transmit an observation when the encoding is ideal for q (instead of p) [108, 98].

Consider two discrete random variables $X \sim p_X$ and $Y \sim p_Y$. The conditional entropy $H[X \mid Y]$ of X given Y is defined as

$$H[X \mid Y] = -\sum_y p_Y(y) \sum_x p(x \mid y) \log_2 p(x \mid y) = \sum_y p_Y(y) H[X \mid Y = y], \tag{2.64}$$

where $0 \log_2 0$ is replaced by 0. It is possible to show that $H[X \mid Y] \leq H[X]$ for any X and Y [108].

The mutual information $I(X; Y)$ between X and Y is defined as

$$I(X; Y) = \mathrm{KL}(p_{X,Y}||p_X p_Y) = \sum_x \sum_y p_{X,Y}(x, y) \log_2 \frac{p_{X,Y}(x, y)}{p_X(x) p_Y(y)}, \tag{2.65}$$

where $p_X(x)$, $p_Y(y)$ and $p_{X,Y}(x, y)$ are non-zero for all x, y. If any of them is zero, the corresponding term inside the summation is again replaced by 0. It is easy to show that [108]

$$I(X; Y) = H[X] - H[X \mid Y] = H[Y] - H[Y \mid X]. \tag{2.66}$$

Intuitively, the mutual information $I(X; Y)$ between X and Y measures the amount of information that Y has about X (dependence between X and Y). Clearly, $I(X; Y) = I(Y; X)$, and $I(X; Y) \geq 0$ [108, 81]. It is also possible to show that $I(X; Y) = 0$ if and only if X and Y are independent [108, 81].

In the context of feature selection for classification tasks, the mutual information $I(X_j; Y)$ can be used to measure how much information a discrete feature variable $X_j \sim$

p_{X_j} has about the class variable $Y \sim p_Y$. This requires approximating the (typically unknown) probability mass functions p_{X_j}, p_Y and $p_{X_j,Y}$ by their corresponding empirical distributions given an iid dataset $\mathcal{D}$. However, these approximations may be unreliable, particularly when the number of observations in $\mathcal{D}$ is small. Furthermore, notice that this approach does not take into account whether a feature is important when taken together with other features.

2.4.2 Randomized logistic regression

Recall that logistic regression is a binary classification technique that assumes that the probability $p(y \mid \mathbf{x}, \mathbf{w})$ of class $y \in \{0, 1\}$ given observation $\mathbf{x}$ and weight vector $\mathbf{w}$ is given by

$$p(y \mid \mathbf{x}, \mathbf{w}) = \sigma(\mathbf{w}\mathbf{x})^y (1 - \sigma(\mathbf{w}\mathbf{x}))^{1-y}, \tag{2.67}$$

where σ is the sigmoid function.

Given an iid dataset $\mathcal{D}$, we have also shown how an appropriate weight vector $\mathbf{w}^*$ can be found by maximizing the log-likelihood $\ell(\mathbf{w}) = \log p(\mathcal{D} \mid \mathbf{w})$ with respect to $\mathbf{w}$. We also mentioned that this maximization objective can lead to a weight vector $\mathbf{w}^*$ with very large coefficients. This leads to a classifier that is very sensitive to small changes in the input observations, which may be overfitted to the training data. This issue can be circumvented by maximizing the l^2-regularized log-likelihood given by $\ell(\mathbf{w}) - \lambda||\mathbf{w}||^2$, for some penalty $\lambda > 0$.

Another alternative is to maximize the l^1-regularized log-likelihood given by $\ell(\mathbf{w}) - \lambda \sum_j |w_j|$. In comparison to the l^2-regularized log-likelihood, this alternative objective favors a sparse solution $\mathbf{w}^*$ (where many elements are zero) [64].

Randomized logistic regression is a feature scoring heuristic based on the idea that if $\mathbf{w}^*$ is the weight vector that maximizes the l^1-regularized log-likelihood, and $w_j^* = 0$, then feature j is not relevant for classification [103]. Notice that this heuristic takes into consideration that a feature may be discriminative when combined with others even if it is not discriminative by itself.

However, notice that $w_j^* = 0$ does not necessarily imply that a feature is not relevant for classification. For instance, a feature k that is always identical to feature j could exist, and w_k^* could be nonzero. Clearly, this example generalizes to groups of features. Although this behavior is appropriate for selecting subsets of features, it is not appropriate for scoring features individually.

Randomized logistic regression deals with this issue by introducing randomness into the learning process. We discuss one possible implementation of the original approach, which is based on stability selection [103].

Consider a sequence of datasets $\mathcal{D}_1, \ldots, \mathcal{D}_S$, each obtained by randomly choosing subsets composed of $N' < N$ elements from an original iid dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$. Suppose a logistic regression classifier is fitted independently for each of these datasets, resulting in a sequence of weight vectors $\mathbf{w}_1^*, \ldots, \mathbf{w}_S^*$. This approach is similar (but not equivalent) to bagging, which we already introduced in Sec. 2.3.4.

The score s_j of feature j is defined as the fraction of classifiers where feature j was

associated to a *significant* coefficient, and is given by

$$s_j = \frac{1}{S} \sum_{i=1}^{S} \mathbb{I}(|w_{i,j}^*| \geq \epsilon), \tag{2.68}$$

where $\mathbb{I}$ is the indicator function, and $\epsilon \geq 0$ is a small constant. Intuitively, the variability introduced by subsampling is likely to affect which features are used by each classifier, avoiding the issue involving related features [103].

So far, we have omitted the main idea in stability selection applied to logistic regression, which is penalizing each feature differently when considering each dataset. This idea can be implemented by maximizing, for each dataset $\mathcal{D}_i$, the l^1-regularized log-likelihood with respect to $\mathbf{w}$ considering randomized penalties, which is given by

$$l(\mathbf{w}) - \lambda \sum_j \frac{|w_j|}{\omega_j}, \tag{2.69}$$

where λ is the usual penalty hyperparameter, but each ω_j is chosen uniformly at random from the set $\{r, 1\}$, where $0 < r < 1$ is a hyperparameter of the feature selection method. Intuitively, this introduces even more variability in the set of features that is likely to be used by each classifier [103].

2.4.3 Recursive feature elimination

Recursive feature elimination (RFE) is a simple greedy heuristic method that attempts to rank features according to their discriminative capacities. The method consists on training a classifier on a dataset $\mathcal{D}$, eliminating the feature(s) with lowest importance according to some criteria, and repeating the process recursively until all features have been eliminated [62]. RFE is an instance of the traditional backward feature selection strategy [62].

In (linear) support vector machine RFE [62], which is a common implementation, the parameters $\mathbf{w}^*$ and b^* of a maximum margin separating hyperplane for the dataset $\mathcal{D}$ are obtained as usual, and the feature j with lowest absolute weight coefficient $|w_j^*|$ is eliminated. The procedure continues recursively without feature j. The score (inverse rank) s_j of feature j is simply the iteration at which it was eliminated.

Similarly to randomized logistic regression, this method assumes that features with small associated coefficients are unimportant for classification. However, SVM RFE does not address the fact that a relevant feature might have been replaced (by the classifier) by a highly related feature (or group of features), as detailed in the previous section. Therefore, it is not appropriate for scoring features individually.

2.4.4 Random forest scoring

A simple heuristic method can be employed to compute the importance of each feature given a classification tree fitted to a dataset [22].

Recall that any vertex u in a classification tree $T = (V, E)$ fitted to a dataset $\mathcal{D}$ is

associated to a region R_u. Define the weighted decrease in cost Δ_u at a non-leaf vertex u as

$$\Delta_u = |\mathcal{D}_u| c(\mathcal{D}_u) - \big[|\mathcal{D}_v| c(\mathcal{D}_v) + |\mathcal{D}_w| c(\mathcal{D}_w)\big], \tag{2.70}$$

where v and w are the children of u in T, $c(\mathcal{D}_z)$ is the cost (impurity) associated to dataset $\mathcal{D}_z$, and $\mathcal{D}_z = \{(\mathbf{x}, y) \in \mathcal{D} \mid \mathbf{x} \in R_z\}$, for $z \in \{u, v, w\}$. In other words, Δ_u is the difference between the cost associated to vertex u, and the combined costs of its two children u and w. Each cost is also weighted by the size of the dataset that reaches the particular vertex during the procedure that builds the tree.

The unnormalized score $\hat{s}_j$ of feature j given a classification tree T fitted to a dataset $\mathcal{D}$ is defined as

$$\hat{s}_j = \sum_{u \in U_j} \Delta_u, \tag{2.71}$$

where $U_j \subseteq V$ is the subset of (non-leaf) vertices in the classification tree T that partition the remaining dataset considering feature j during the procedure that builds the tree. In other words, for any $u \in U_j$ with children v and w, $R_v = \{\mathbf{x} \in R_u \mid x_j < \tau\}$, and $R_w = \{\mathbf{x} \in R_u \mid x_j \geq \tau\}$, for some threshold τ.

Intuitively, features with high unnormalized scores contributed significantly to the decrease in impurity, possibly while the procedure that builds the tree was still considering a large remaining dataset.

It is usually more convenient to consider the normalized score s_j of feature j, which is given by

$$s_j = \frac{\hat{s}_j}{\sum_k \hat{s}_k}. \tag{2.72}$$

A random forest classifier [23] or extremely randomized tree [51] may compute feature scores by averaging the normalized scores given by the individual trees that comprise the ensemble. As in randomized logistic regression, this heuristic attempts to avoid associating a high score to a single important feature in detriment of other highly related features.

Notice that these feature scoring methods based on classification trees are naturally capable of dealing with any number of classes, and do not suppose that the decision boundary is an affine hyperplane, in contrast to both randomized logistic regression and SVM recursive feature elimination.

2.5 High-dimensional data visualization

High-dimensional data visualization is a subarea of data visualization that is concerned with creating visual depictions of high-dimensional datasets. For an extensive overview of the field, we refer to the recent survey by Liu *et al.* [96].

There are many alternatives for visual exploration of high-dimensional data, such as parallel coordinate plots [67], radial layouts [69, 77, 90], table lenses [121], and scatterplot matrices [10]. A common challenge for these methods is scalability to datasets with relatively modest numbers of observations *and* dimensions, as will become clear in the next sections. We introduce dimensionality reduction as an alternative to these methods in Sec. 2.6.

2.5.1 Table lenses

Consider a dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$, such that $\mathbf{x}_i \in \mathbb{R}^D$, and its corresponding $N \times D$ design matrix $\mathbf{X}$, where each observation corresponds to a row. A natural way to visualize such dataset is to represent the design matrix by a traditional numeric table.

The effectiveness of a simple numeric table can be greatly enhanced by basic interactivity, such as allowing the user to sort groups of observations (rows) by a specific feature (column). Instead of representing the feature values using numbers, it is also possible to use a visual metaphor, such as an appropriately scaled (and possibly colored) bar. Through *zoom* and *panning*, the user becomes able to visualize a larger number of observations at once, at an appropriate level of detail. A combination of these and related improvements leads to an interactive visualization method called table lens [121, 142] (see Fig. 2.8).

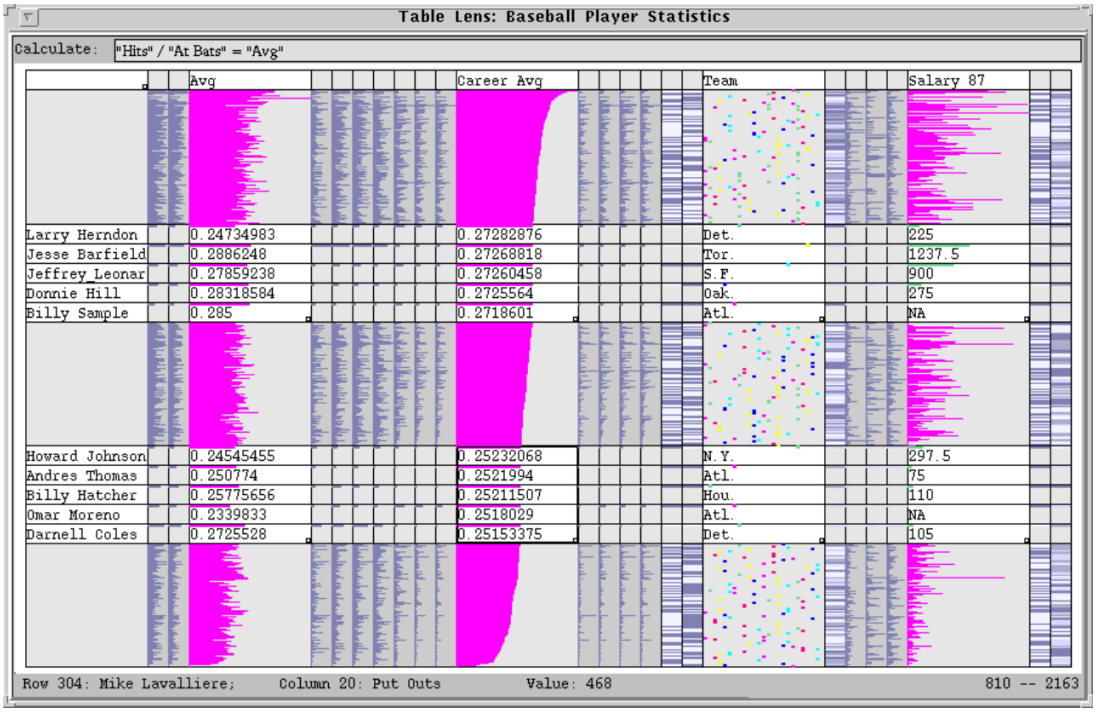


Figure 2.8: Table lens showing baseball player statistics. Source: [121].

Although this method is scalable with respect to the number of observations, it provides limited insight about complex structures that involve several features, such as clusters (groups of similar observations). In this sense, it has limited scalability with respect to the number of features. Its effectiveness is also highly dependent on the ordering of the columns. For instance, even simple relationships (*e.g.*, linear correlation) between features can be obscured when the corresponding columns are placed far apart.

2.5.2 Scatterplot matrices

Consider a dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$, such that $\mathbf{x}_i \in \mathbb{R}^D$. A scatterplot matrix is a $D \times D$ table $\mathbf{T}$ composed of scatterplots. Each scatterplot $T_{i,j}$ shows each observation in $\mathcal{D}$

restricted to a pair of features $i, j \in \{1, \ldots, D\}$ in Cartesian coordinates [10] (see Fig. 2.9). The scatterplots in the diagonal of $\mathbf{T}$ are often substituted by feature histograms.

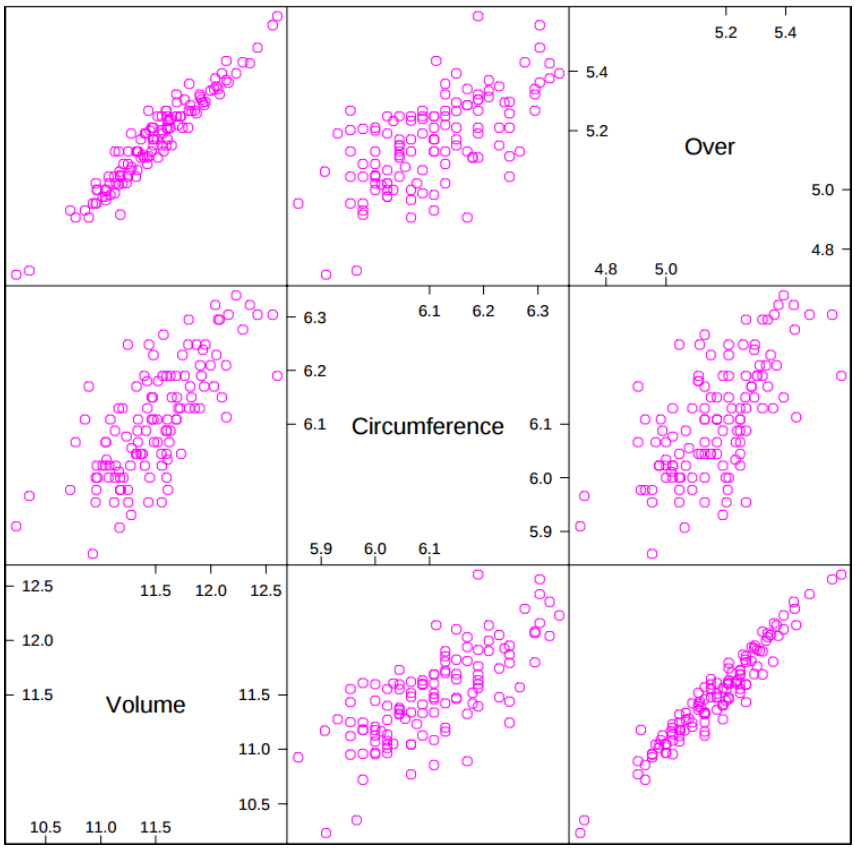


Figure 2.9: Scatterplot matrix showing a three-dimensional dataset. Source: [10].

This visualization method allows detecting some types of clusters and dependence between features. However, it is incapable of (directly) representing relationships between more than two features, and its scalability with respect to the number of features is severely limited.

2.5.3 Parallel coordinate plots

Consider a dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$, such that $\mathbf{x}_i \in \mathbb{R}^D$. A parallel coordinate plot is composed of D parallel line segments (axes), typically positioned vertically and equally spaced [67] (see Fig. 2.10). Each of these axes represents the range of values of a single feature. Furthermore, each observation $\mathbf{x}_i$ is represented by a polyline (sequence of connected line segments) whose vertex j intersects the axis corresponding to feature j precisely at the position that represents the value $x_{i,j}$.

A parallel coordinate plot is useful to detect the presence of clusters or outliers, and also to detect dependence between features. However, the effectiveness of a parallel coordinate plot is highly dependent on the (visual) order of the features, and the method does not scale well to more than a few dozen features.

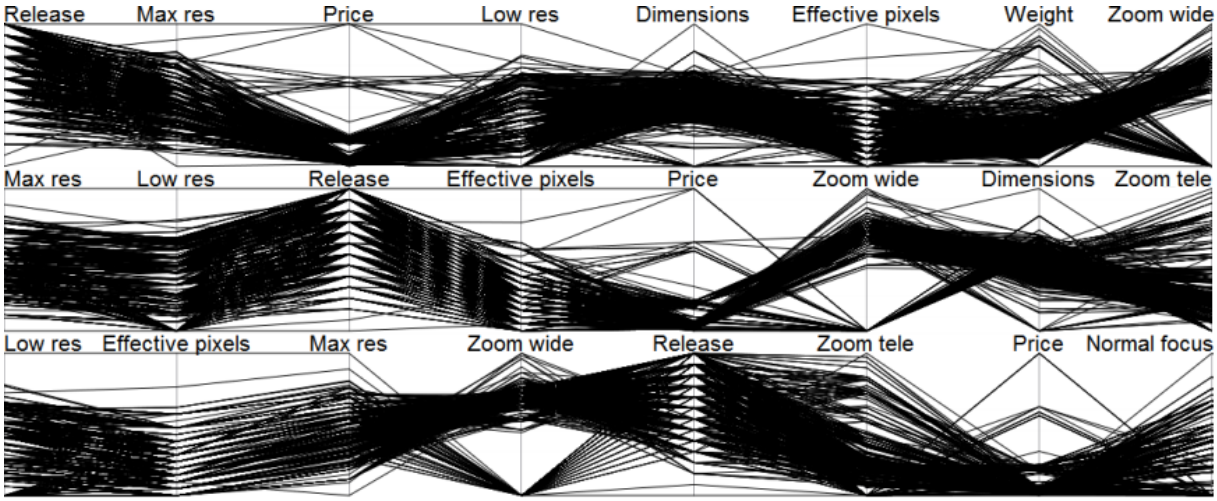


Figure 2.10: Parallel coordinates showing three different orderings of features. Source: [66] (adapted).

2.6 Dimensionality reduction for visualization

Dimensionality reduction (DR) techniques effectively address the scalability issues of the previously discussed high-dimensional data visualization techniques by finding a low-dimensional representation of the data that retains *structure*, which is defined by relationships between points, presence of clusters, or overall spatial data distribution [95, 96, 88, 148]. In this text, we refer to the representation obtained by DR by the term *projection*. For the purposes of visualization, DR techniques typically reduce the number of dimensions to two or three.

More concretely, dimensionality reduction techniques typically address the task of representing an iid dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$, where $\mathbf{x}_i \in \mathbb{R}^D$, by a projection $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$, where $\mathbf{p}_i \in \mathbb{R}^d$, $d < D$, and each *point* $\mathbf{p}_i$ corresponds to observation $\mathbf{x}_i$. Different techniques attempt to preserve different aspects of the dataset $\mathcal{D}$ in the projection $\mathcal{P}$.

The resulting projections can be represented as scatterplots, which allow reasoning about clusters, outliers, and trends by direct visual exploration. These and other tasks addressed by DR-based visualizations are detailed by Brehmer *et al.* [21]. Visual exploration of high-dimensional datasets via projections has been widely applied to many types of data, such as text documents [116], multimedia collections [75], gene expressions [24], and networks [25].

DR techniques are typically divided into linear (*e.g.*, PCA, LDA, classical MDS) and non-linear (*e.g.*, Isomap, LLE, t-SNE) [30, 88, 148], based on the properties of the mapping from $\mathbb{R}^D$ to $\mathbb{R}^d$. Although many traditional DR techniques are computationally expensive, highly scalable techniques have also been proposed (*e.g.*, LSP [116], LAMP [75], LoCH [40]). These techniques are currently capable of dealing with hundreds of thousands of high-dimensional observations (or more), although visual clutter eventually becomes a problem. Guidelines for choosing suitable DR methods for a particular task are outlined by Sedlmair *et al.* [133]. In Chapter 4, we will introduce several visualization techniques that have been proposed to help the interactive exploration of projections.

The next sections describe four widespread dimensionality reduction techniques in detail: principal component analysis (PCA), linear discriminant analysis (LDA), multidimensional scaling (MDS), and t-distributed stochastic neighbor embedding (t-SNE). Although PCA and LDA are not employed in the next chapters, presenting these techniques is justified by their popularity. LDA is particularly interesting as the only supervised dimensionality reduction technique that we discuss. MDS and t-SNE will be employed in Chapters 4 and 5.

Finally, consider the task of visualizing a sequence of datasets that represents a time-dependent process using dimensionality reduction. If a DR technique is applied independently for each time step, the resulting sequence of projections may present variability that does not reflect significant changes in the *structure* of the data. We refer to this issue as *temporal incoherence*, which significantly impairs the visualization of temporal trends. Temporal incoherence will affect any DR technique that is sensitive to relatively small changes in their inputs [49]. We address this issue in Chapter 6, focusing on t-SNE.

2.6.1 Principal component analysis

Principal component analysis (PCA) is a widely used dimensionality reduction technique [30, 15]. Consider a dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$. Firstly, notice that the mean projection onto the vector $\mathbf{u}_1 \in \mathbb{R}^D$ of the observations in $\mathcal{D}$ can be written as

$$\frac{1}{N}\sum_{i=1}^{N}\mathbf{u}_1\mathbf{x}_i = \mathbf{u}_1\frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i = \mathbf{u}_1\bar{\mathbf{x}}, \tag{2.73}$$

where $\bar{\mathbf{x}}$ is the empirical mean. Consider the task of finding an *unit* vector $\mathbf{u}_1$ such that the variance

$$v(\mathbf{u}_1) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{u}_1\mathbf{x}_i - \mathbf{u}_1\bar{\mathbf{x}})^2 \tag{2.74}$$

of the projections onto $\mathbf{u}_1$ of the observations in $\mathcal{D}$ is maximum. It can be easily shown that $v(\mathbf{u}_1)$ can also be written as

$$v(\mathbf{u}_1) = \frac{1}{N}\sum_{i=1}^{N}\left[(\mathbf{u}_1\mathbf{x}_i)^2 - 2(\mathbf{u}_1\mathbf{x}_i)(\mathbf{u}_1\bar{\mathbf{x}}) + (\mathbf{u}_1\bar{\mathbf{x}})^2\right] = \mathbf{u}_1^T\boldsymbol{\Sigma}\mathbf{u}_1, \tag{2.75}$$

where $\boldsymbol{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T$ is the empirical covariance matrix of the dataset $\mathcal{D}$.

The maximization task outlined above corresponds to finding the maxima of v restricted to the D-dimensional unit sphere $S = \{\mathbf{u}_1 \in \mathbb{R}^D \mid g(\mathbf{u}_1) = 0\}$, where $g(\mathbf{u}_1) = 1 - \mathbf{u}_1\mathbf{u}_1$. The Lagrange multiplier theorem states that if $\mathbf{u}_1$ is a local maximum of v restricted to S, and $\nabla g(\mathbf{u}_1) \neq 0$, then

$$\nabla v(\mathbf{u}_1) + \lambda_1 \nabla g(\mathbf{u}_1) = 0, \tag{2.76}$$

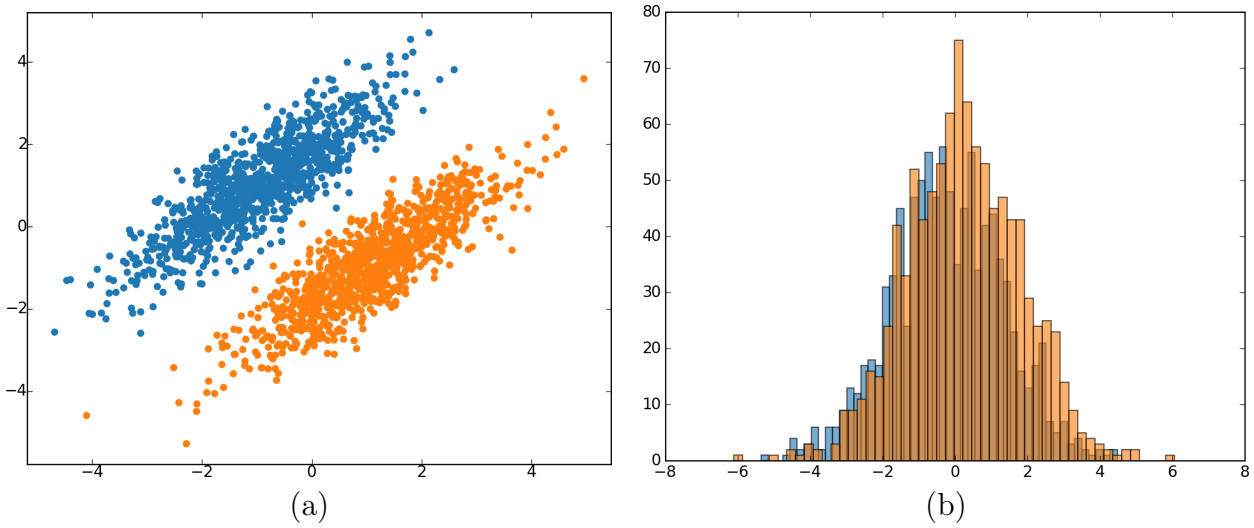


Figure 2.11: (a) Original 2D dataset. (b) 1D PCA projection represented by one semi-transparent histogram per class. Notice the poor separation between observations from distinct classes in the projection.

for some $\lambda_1 \in \mathbb{R}$. Thus,

$$0 = \nabla_{\mathbf{u}_1}\left[\mathbf{u}_1^T \boldsymbol{\Sigma} \mathbf{u}_1\right] + \lambda_1 \nabla_{\mathbf{u}_1}\left[1 - \mathbf{u}_1 \mathbf{u}_1\right] \tag{2.77}$$

$$= (\boldsymbol{\Sigma} + \boldsymbol{\Sigma}^T)\mathbf{u}_1 - 2\lambda_1 \mathbf{u}_1 = 2\boldsymbol{\Sigma}\mathbf{u}_1 - 2\lambda_1 \mathbf{u}_1, \tag{2.78}$$

where $(\boldsymbol{\Sigma} + \boldsymbol{\Sigma}^T) = 2\boldsymbol{\Sigma}$ because $\boldsymbol{\Sigma}$ is symmetric. Therefore, if $\mathbf{u}_1$ is a local maximum of v restricted to S, then $\boldsymbol{\Sigma}\mathbf{u}_1 = \lambda_1 \mathbf{u}_1$ for some λ_1. In other words, $\mathbf{u}_1$ is an eigenvector of $\boldsymbol{\Sigma}$ with eigenvalue λ_1. Because the variance $v(\mathbf{u}_1)$ corresponding to such an eigenvector $\mathbf{u}_1$ is the corresponding eigenvalue $\lambda_1 = \mathbf{u}_1^T \boldsymbol{\Sigma} \mathbf{u}_1$, the eigenvector with maximum associated eigenvalue is the desired global maximum restricted to S, although we will not show that this condition is indeed sufficient.

Consider an orthonormal list $L_i = (\mathbf{u}_1, \ldots \mathbf{u}_i)$ containing the *largest* eigenvectors (with respect to their corresponding eigenvalues) of an empirical covariance matrix $\boldsymbol{\Sigma}$, sorted in non-increasing order, for some $1 \leq i \leq D$. It can be shown by induction that $\mathbf{u}_j$ is a vector with maximum variance $v(\mathbf{u}_j)$ among all unit vectors that are orthogonal to all the vectors in the list $(\mathbf{u}_1, \ldots, \mathbf{u}_{j-1})$ [15].

Using such a list of eigenvectors L_d, any point $\mathbf{x}_i \in \mathbb{R}^D$ can be represented by a point $\mathbf{p}_i = (\mathbf{u}_1 \mathbf{x}_i, \ldots, \mathbf{u}_d \mathbf{x}_i)$ in $\mathbb{R}^d$. Therefore, PCA performs a linear mapping between $\mathbb{R}^D$ and $\mathbb{R}^d$. For an illustrative example, see Fig. 2.11.

In practice, it is essential to transform an original dataset $\mathcal{D}'$ into a dataset $\mathcal{D}$ with empirical mean $\bar{\mathbf{x}} = 0$ before applying principal component analysis, since the increase in variance along a direction due to translations is usually irrelevant. In that case, principal component analysis also finds the linear map with minimum reconstruction error [30]. In many applications, the dataset $\mathcal{D}$ should also have the same empirical variance across each feature (i.e., the dataset $\mathcal{D}'$ should be standardized) to minimize the effect of the choice of units.

Projections obtained by principal component analysis may fail to preserve structures of interest. For instance, Fig. 2.11 presents a case where two evident clusters are merged in a 1D projection. Analogously to pattern classification, because each technique is concerned with preserving different aspects of a dataset in its projection, there is no single *best* dimensionality reduction technique.

2.6.2 Linear discriminant analysis

Linear discriminant analysis can be used as a dimensionality reduction technique that takes class information into account [108, 64]. This may result in projections where the classes are better separated when compared to projections obtained by techniques that ignore such information.

We present linear discriminant analysis from a slightly unusual perspective, which is mostly based on [108] and [64]. Although the technique was originally proposed as the solution to finding a linear map that maximizes inter-class variance while minimizing intra-class variance, our presentation explicits its underlying assumptions. We refer to [64] for further details.

The multivariate Gaussian joint probability density function $\mathcal{N}(\cdot \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ is defined on $\mathbb{R}^D$ as

$$\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^D |\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \tag{2.79}$$

where $\boldsymbol{\mu} \in \mathbb{R}^D$ is the mean vector and $\boldsymbol{\Sigma}$ is the $D \times D$ (positive definite) covariance matrix. Indeed, if $\mathbf{X} \sim \mathcal{N}(\cdot \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$, then $\mathbb{E}[\mathbf{X}] = \boldsymbol{\mu}$ and $\mathrm{cov}[\mathbf{X}] = \boldsymbol{\Sigma}$.

Consider the task of creating a classifier given the dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, which is iid according to $p(\cdot \mid \boldsymbol{\theta}^*)$, for an unknown $\boldsymbol{\theta}^*$. Consider also that $\mathbf{x}_i \in \mathbb{R}^D$, and $y_i \in \{1, \ldots C\}$.

Gaussian discriminant analysis assumes that the density $p(\mathbf{x} \mid y, \boldsymbol{\theta})$ associated to observation $\mathbf{x}$ given the class y and the parameter vector $\boldsymbol{\theta}$ is given by [108]

$$p(\mathbf{x} \mid y, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y). \tag{2.80}$$

Note that $\boldsymbol{\mu}_y$ and $\boldsymbol{\Sigma}_y$ are represented in $\boldsymbol{\theta}$, for every y. In words, the technique assumes that the observations in each class are distributed according to distinct multivariate Gaussian distributions.

For a particular estimate $\hat{\boldsymbol{\theta}}$ of the parameters (maximum likelihood, for instance), classification uses the fact that

$$p(y \mid \mathbf{x}, \hat{\boldsymbol{\theta}}) = \frac{p(\mathbf{x} \mid y, \hat{\boldsymbol{\theta}}) p(y \mid \hat{\boldsymbol{\theta}})}{p(\mathbf{x} \mid \hat{\boldsymbol{\theta}})} \propto_y p(\mathbf{x} \mid y, \hat{\boldsymbol{\theta}}) p(y \mid \hat{\boldsymbol{\theta}}) = \mathcal{N}(\mathbf{x} \mid \hat{\boldsymbol{\mu}}_y, \hat{\boldsymbol{\Sigma}}_y) \hat{\pi}_y, \tag{2.81}$$

for all $\mathbf{x}$ and y. Note that $\hat{\pi}_y = p(y \mid \hat{\boldsymbol{\theta}})$ is also represented in $\hat{\boldsymbol{\theta}}$, for every y. The symbol $\propto_y$ denotes proportionality with respect to variable y.

Consider the task of finding the maximum (log-)likelihood estimate for Gaussian dis-

criminant analysis. By definition, the log-likelihood $\log p(\mathcal{D} \mid \boldsymbol{\theta})$ of $\boldsymbol{\theta}$ given $\mathcal{D}$ is

$$\log \prod_{i=1}^{N} p(\mathbf{x}_i \mid y_i, \boldsymbol{\theta}) p(y_i \mid \boldsymbol{\theta}) = \sum_{y=1}^{C} \sum_{i=1|y_i=y}^{N} \log \mathcal{N}(\mathbf{x}_i \mid \boldsymbol{\mu}_y, \boldsymbol{\Sigma}_y) + \sum_{y=1}^{C} N_y \log \pi_y. \tag{2.82}$$

Because the first summation can be maximized (with respect to the mean vectors and covariance matrices) independently for each class, the maximum likelihood estimates are simply given by [108]

$$\hat{\boldsymbol{\mu}}_y = \frac{1}{N_y} \sum_{i=1|y_i=y}^{N} \mathbf{x}_i \tag{2.83}$$

$$\hat{\boldsymbol{\Sigma}}_y = \frac{1}{N_y} \sum_{i=1|y_i=y}^{N} (\mathbf{x}_i - \hat{\boldsymbol{\mu}}_y)(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_y)^T \tag{2.84}$$

$$\hat{\pi}_y = \frac{N_y}{N}, \tag{2.85}$$

where N_y is the number of observations in $\mathcal{D}$ that belong to class y. We say $\hat{\boldsymbol{\mu}}_y$ is the class y centroid. If there are insufficient observations in class y, the covariance matrix estimate $\hat{\boldsymbol{\Sigma}}_y$ may be non-invertible, and thus invalid as a covariance matrix. Even if $\hat{\boldsymbol{\Sigma}}_y$ is invertible, it may overfit the data.

Linear discriminant analysis (LDA) addresses this issue by assuming that the covariance matrix is the same for all classes [64, 108]. Thus, for a particular estimate $\hat{\boldsymbol{\theta}}$,

$$p(y \mid \mathbf{x}, \hat{\boldsymbol{\theta}}) \propto_y \hat{\pi}_y \mathcal{N}(\mathbf{x} \mid \hat{\boldsymbol{\mu}}_y, \hat{\boldsymbol{\Sigma}}) \propto_y \hat{\pi}_y \exp\left[-\frac{1}{2}(\mathbf{x} - \hat{\boldsymbol{\mu}}_y)^T \hat{\boldsymbol{\Sigma}}^{-1} (\mathbf{x} - \hat{\boldsymbol{\mu}}_y) \right], \tag{2.86}$$

for all $\mathbf{x}$ and y. Using the distributivity of matrix multiplication over addition,

$$p(y \mid \mathbf{x}, \hat{\boldsymbol{\theta}}) \propto_y \hat{\pi}_y \exp\left[\hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{x} - \frac{1}{2} \hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_y - \frac{1}{2} \mathbf{x}^T \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{x} \right] \tag{2.87}$$

$$= e^{\log \hat{\pi}_y} \exp\left[\hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{x} - \frac{1}{2} \hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_y \right] \exp\left[-\frac{1}{2} \mathbf{x}^T \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{x} \right] \tag{2.88}$$

$$\propto_y \exp\left[\hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1} \mathbf{x} - \frac{1}{2} \hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_y + \log \hat{\pi}_y \right]. \tag{2.89}$$

Letting $\boldsymbol{\beta}_y = \hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1}$ and $\gamma_y = -\frac{1}{2} \hat{\boldsymbol{\mu}}_y^T \hat{\boldsymbol{\Sigma}}^{-1} \hat{\boldsymbol{\mu}}_y + \log \hat{\pi}_y$,

$$p(y \mid \mathbf{x}, \hat{\boldsymbol{\theta}}) = \frac{e^{\boldsymbol{\beta}_y \mathbf{x} + \gamma_y}}{\sum_{y'=1}^{C} e^{\boldsymbol{\beta}_{y'} \mathbf{x} + \gamma_{y'}}}, \tag{2.90}$$

since $\sum_y p(y \mid \mathbf{x}, \hat{\boldsymbol{\theta}}) = 1$, for any $\mathbf{x}$ and $\hat{\boldsymbol{\theta}}$.

Consider the set S composed of all $\mathbf{x}$ such that $p(y \mid \mathbf{x}, \hat{\boldsymbol{\theta}}) = p(y' \mid \mathbf{x}, \hat{\boldsymbol{\theta}})$, for a particular choice of $y \neq y'$ and $\hat{\boldsymbol{\theta}}$. Clearly, $e^{\boldsymbol{\beta}_y \mathbf{x} + \gamma_y} = e^{\boldsymbol{\beta}_{y'} \mathbf{x} + \gamma_{y'}}$, and $\boldsymbol{\beta}_y \mathbf{x} + \gamma_y = \boldsymbol{\beta}_{y'} \mathbf{x} + \gamma_{y'}$. Rearranging the terms, $(\boldsymbol{\beta}_y - \boldsymbol{\beta}_{y'})\mathbf{x} = \gamma_{y'} - \gamma_y$. Thus, the decision boundary S between

any such y and y' is an affine hyperplane, which originates the term linear discriminant analysis [64].

An appropriate covariance matrix estimate $\hat{\boldsymbol{\Sigma}}$ is given by [64]

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{N-C}\sum_{i=1}^{N}(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{y_i})(\mathbf{x}_i - \hat{\boldsymbol{\mu}}_{y_i})^T. \tag{2.91}$$

Suppose such covariance matrix estimate $\hat{\boldsymbol{\Sigma}}$ obtained from a dataset $\mathcal{D}$ is the identity matrix $\mathbf{I}$. In this case, because $\hat{\boldsymbol{\Sigma}} = \hat{\boldsymbol{\Sigma}}^{-1} = \mathbf{I}$,

$$p(y \mid \mathbf{x}, \hat{\boldsymbol{\theta}}) \propto_y \exp\left[\hat{\boldsymbol{\mu}}_y\mathbf{x} - \frac{1}{2}||\hat{\boldsymbol{\mu}}_y||^2 + \log\hat{\pi}_y\right]. \tag{2.92}$$

Recall that the squared Euclidean distance between the centroid $\hat{\boldsymbol{\mu}}_y$ and observation $\mathbf{x}$ is given by $||\hat{\boldsymbol{\mu}}_y - \mathbf{x}||^2 = ||\hat{\boldsymbol{\mu}}_y||^2 + ||\mathbf{x}||^2 - 2\hat{\boldsymbol{\mu}}_y\mathbf{x}$. Therefore, maximizing $\hat{\boldsymbol{\mu}}_y\mathbf{x} - \frac{1}{2}||\hat{\boldsymbol{\mu}}_y||^2$ with respect to y corresponds to minimizing $||\hat{\boldsymbol{\mu}}_y - \mathbf{x}||^2$. In other words, in the case of an identity covariance matrix and uniform prior class probabilities, classifying an observation by linear discriminant analysis corresponds to finding the class with the closest centroid [64].

The idea outlined in the previous paragraph is at the core of dimensionality reduction based on linear discriminant analysis [108, 64]. Next, we introduce the whitening transform, which will be required to obtain a dataset with the desired identity covariance matrix.

Let $\mathbf{X}$ be a D-dimensional random vector such that $\boldsymbol{\Sigma} = \text{cov}[\mathbf{X}]$ is positive definite and $\mathbb{E}[\mathbf{X}] = \mathbf{0}$. Consider the eigendecomposition $\boldsymbol{\Sigma} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$, where each column j of the $D \times D$ matrix $\boldsymbol{U}$ is an eigenvector $\mathbf{u}_j$ of $\boldsymbol{\Sigma}$, and $\boldsymbol{\Lambda}$ is a diagonal matrix such that $\Lambda_{j,j} = \lambda_j$ is the eigenvalue that corresponds to $\mathbf{u}_j$. Furthermore, $\boldsymbol{U}$ is chosen so that $\boldsymbol{U}^T\boldsymbol{U} = \boldsymbol{I}$, which means that $(\mathbf{u}_1, \ldots, \mathbf{u}_D)$ is an orthonormal basis for $\mathbb{R}^D$. Let $\boldsymbol{\Lambda}^{-\frac{1}{2}}$ denote the diagonal matrix such that $\boldsymbol{\Lambda}_{j,j}^{-\frac{1}{2}} = 1/\sqrt{\lambda_j}$. It can be shown that if $\mathbf{W} = \boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{U}^T\mathbf{X}$, then $\mathbb{E}[\mathbf{W}] = \mathbf{0}$ and $\text{cov}[\mathbf{W}] = \mathbf{I}$ [64]. In other words, the so-called whitening matrix $\boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{U}^T$ maps a random vector $\mathbf{X}$ to a column matrix $\mathbf{W}$ (which may be seen as a random vector) whose covariance matrix is the identity.

Consider a *centered* iid dataset $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, such that $\sum_i \mathbf{x}_i = \mathbf{0}$. Let $\hat{\boldsymbol{\Sigma}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$ denote the eigendecomposition of the covariance matrix estimate given by linear discriminant analysis (Eq. 2.91). If we suppose such estimate is correct, then the corresponding covariance matrix estimate for the whitened dataset $\mathcal{D}' = \{(\boldsymbol{\Lambda}^{-\frac{1}{2}}\boldsymbol{U}^T\mathbf{x}, y) \mid (\mathbf{x}, y) \in \mathcal{D}\}$ should be the identity matrix [64]. As a consequence, a transformed observation can be classified according to the closest transformed centroid (assuming uniform prior class probabilities).

Let $\mathcal{D}'_{\boldsymbol{\mu}} = \boldsymbol{\mu}'_1, \ldots, \boldsymbol{\mu}'_C$ denote a dataset composed of the transformed centroids. Note that this sequence of vectors is not linearly independent, since $\sum_y a_y\boldsymbol{\mu}'_y = \mathbf{0}$, for some $a_y > 0$, which follows from the fact that $\mathcal{D}$ is centered. As a consequence, they span a subspace of dimension at most $C - 1$.

The main step in dimensionality reduction based on linear discriminant analysis is to

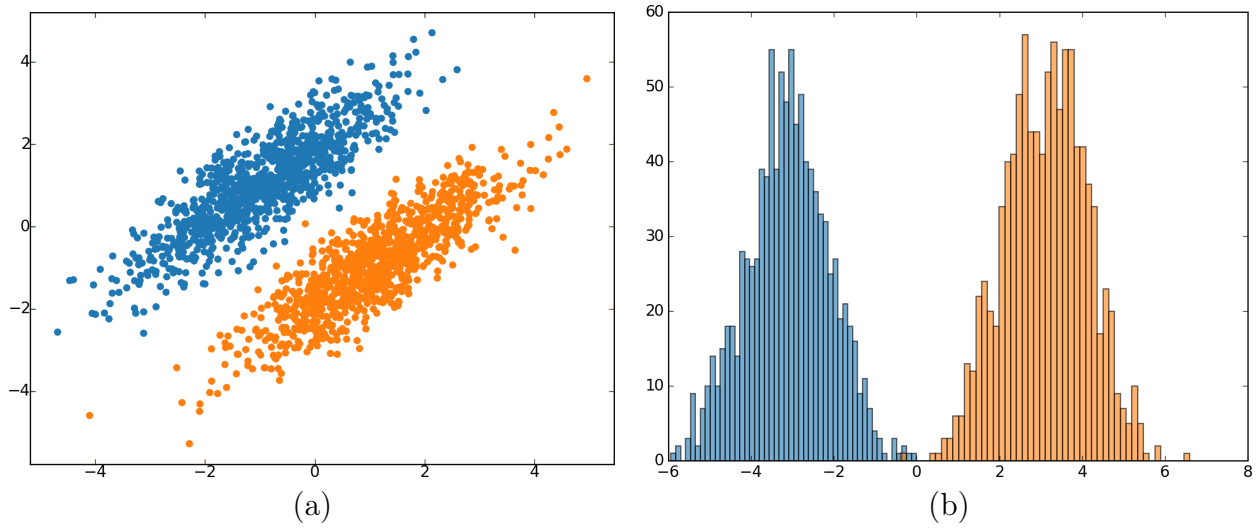


Figure 2.12: (a) Original 2D dataset. (b) 1D LDA projection represented by one semi-transparent histogram per class. Notice the good separation between observations from distinct classes in the projection.

perform principal component analysis on the transformed centroids $\mathcal{D}'_{\boldsymbol{\mu}}$, which results in an orthonormal list $L_{C'} = (\mathbf{u}_1, \ldots \mathbf{u}_{C'})$ composed of $C' < C$ vectors [64]. Projecting the transformed centroids onto these directions leads to maximum variance in a very specific sense, as already explained in the previous section. Intuitively, projecting the transformed observations onto these directions may preserve class separation better than unsupervised dimensionality reduction. For an illustrative example, see Fig. 2.12.

Finally, each observation $\mathbf{x}_i \in \mathbb{R}^D$ in the original (centered) dataset $\mathcal{D}$ can be represented by the point $\mathbf{p}_i \in \mathbb{R}^{C'}$ given by

$$\mathbf{p}_i = (\mathbf{u}_1^T \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{U}^T \mathbf{x}_i, \ldots, \mathbf{u}_{C'}^T \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{U}^T \mathbf{x}_i), \tag{2.93}$$

where $C' < C$. If $C > 2$, the first $d = 2$ elements of $\mathbf{p}_i$ are typically chosen for the purposes of visualization [64]. Notice that dimensionality reduction by linear discriminant analysis is also a linear mapping from $\mathbb{R}^D$ to $\mathbb{R}^{C'}$.

In contrast to PCA (Fig. 2.11), a 1D projection obtained by LDA is capable of preserving the clusters in the example illustrated by Fig. 2.12 (using cluster information). However, it should be clear that such separation between clusters is not guaranteed in the general case.

2.6.3 Multidimensional scaling

Multidimensional scaling (MDS) methods attempt to represent dissimilarities between pairs of objects of interest by distances between points placed in a low-dimensional space [17].

Consider a dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ composed of D-dimensional observations. The goal of *absolute* (metric) multidimensional scaling [17] is to compute a projection $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$ where the distances between observations in $\mathcal{D}$ are preserved, considering that

each $\mathbf{p}_i \in \mathbb{R}^d$ corresponds to an observation $\mathbf{x}_i \in \mathbb{R}^D$.

Let $d_{i,j} = ||\mathbf{x}_i - \mathbf{x}_j||$ denote the Euclidean distance between observations $\mathbf{x}_i$ and $\mathbf{x}_j$. Analogously, let $r_{i,j} = ||\mathbf{p}_i - \mathbf{p}_j||$.

The goal of absolute multidimensional scaling may be achieved by minimizing the so-called (raw) stress cost C with respect to the projection $\mathcal{P}$, which is given by [17]

$$C = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} (d_{i,j} - r_{i,j})^2. \tag{2.94}$$

Intuitively, mismatch between the corresponding distances in the two spaces is penalized.

Highly specialized methods have been applied to optimize this and other multidimensional scaling objectives [17], which are out of our scope. Many MDS variants also admit dissimilarities between objects of interest, in contrast to distances in the strict sense [17].

2.6.4 T-distributed stochastic neighbor embedding

The goal of t-distributed stochastic neighbor embedding (t-SNE) is to compute a projection $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$ where the *neighborhoods* from a dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ are preserved [148], considering that each $\mathbf{p}_i \in \mathbb{R}^d$ corresponds to $\mathbf{x}_i \in \mathbb{R}^D$.

Once again, we will let $d_{i,j} = ||\mathbf{x}_i - \mathbf{x}_j||$ denote the Euclidean distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. Analogously, $r_{i,j} = ||\mathbf{p}_i - \mathbf{p}_j||$.

Firstly, consider a random process where observations are *visited* in sequence. Furthermore, let the probability $P(X' = j \mid X = i)$ of choosing the *next* observation $\mathbf{x}_j$ given the *current* observation $\mathbf{x}_i$ be given by

$$P(X' = j \mid X = i) = \frac{\exp\left(-\frac{d_{i,j}^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{d_{i,k}^2}{2\sigma_i^2}\right)}, \tag{2.95}$$

except for $i = j$, when $P(X' = j \mid X = i) = 0$.

Each parameter $\sigma_i > 0$ is chosen in such a way that the (conditional) *perplexity* $\kappa = 2^{H[X'|X=i]}$ matches a pre-defined value, where $H[X]$ denotes the entropy of X. This is typically accomplished by binary search [148]. As an intuitive aid, notice that an uniformly distributed discrete random variable X that admits K distinct assignments has perplexity $2^{H[X]} = K$. In simplified terms, $P(X' = j \mid X = i)$ is high whenever $\mathbf{x}_j$ is near $\mathbf{x}_i$ relative to the *observation density* near $\mathbf{x}_i$.

Consider also a distinct random process where the probability $P(X' = i, X = j)$ of *choosing* a pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{D} \times \mathcal{D}$ is given by

$$P(X' = i, X = j) = \frac{P(X' = j \mid X = i) + P(X' = i \mid X = j)}{2N}. \tag{2.96}$$

Intuitively, $P(X' = i, X = j)$ is high whenever $P(X' = j \mid X = i)$ or $P(X' = i \mid X = j)$ is high.

In $\mathbb{R}^d$, the probability $P(Y' = i, Y = j)$ of *choosing* a pair $(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{P} \times \mathcal{P}$ in yet

another random process is given by

$$P(Y' = i, Y = j) = \frac{(1 + r_{i,j}^2)^{-1}}{\sum_k \sum_{l \neq k} (1 + r_{k,l}^2)^{-1}}, \tag{2.97}$$

except for $i = j$, when $P(Y' = i, Y = j) = 0$. Clearly, $P(Y' = i, Y = j)$ is high whenever $\mathbf{p}_i$ and $\mathbf{p}_j$ are close.

T-SNE aims at minimizing the Kullback-Leibler divergence C between $P(X', X)$ and $P(Y', Y)$ with respect to $\mathcal{P}$, which is given by

$$C = \sum_i \sum_{j \neq i} P(X' = i, X = j) \log \left[\frac{P(X' = i, X = j)}{P(Y' = i, Y = j)} \right]. \tag{2.98}$$

Recall from Section 2.4.1 that such Kullback-Leibler divergence can be interpreted, in a very specific setting, as the increase in the average number of bits required to transmit an assignment to X' and X when the encoding is ideal for $P(Y', Y)$ instead of $P(X', X)$. For our purposes, it suffices to notice that C heavily penalizes $P(X' = i, X = j) \gg P(Y' = i, Y = j)$ for some i and j, which corresponds to placing *neighbors* in $\mathcal{D}$ far apart in $\mathcal{P}$. In analogy with data compression, this corresponds to associating long codes to frequently occurring symbols. For the converse, notice that associating short codes to infrequent symbols is only an issue insofar as it prevents frequent symbols from having even shorter codes.

The cost C is usually minimized with respect to $\mathcal{P}$ by (momentum-based) gradient descent [148]: from an arbitrary initial $\mathcal{P}$, for a number of iterations, each $\mathbf{p}_i \in \mathcal{P}$ is moved in the direction $-\nabla_{\mathbf{p}_i} C$.

The gradient $\nabla_{\mathbf{p}_i} C$ of C with respect to a point $\mathbf{p}_i \in \mathcal{P}$ is given by

$$\nabla_{\mathbf{p}_i} C = 4 \sum_j (\mathbf{p}_i - \mathbf{p}_j) \frac{P(X' = i, X = j) - P(Y' = i, Y = j)}{1 + r_{i,j}^2}. \tag{2.99}$$

Geometrically, $\nabla_{\mathbf{p}_i} C$ is a combination of vectors pointing in the direction $\mathbf{p}_i - \mathbf{p}_j$, for every j. Each vector $\mathbf{p}_i - \mathbf{p}_j$ is also weighted by whether $\mathbf{p}_j$ should be moved closer to $\mathbf{p}_i$ to preserve neighborhoods from $\mathcal{D}$, and by whether $\mathbf{p}_j$ is close to $\mathbf{p}_i$.

T-SNE can be considered a state-of-the-art dimensionality reduction technique [148, 146], and is widely employed in following chapters. As a technique that attempts to preserve neighborhoods, it benefits from not having to preserve (large) distances particularly well, while still being appropriate for reasoning about groups of similar observations (clusters). However, as we already mentioned, there is no single *best* dimensionality reduction technique, and our work involving projections is mostly independent of choosing t-SNE.

2.6.5 Visualizing projections

The previous sections described several ways to obtain a d-dimensional projection $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$, where $\mathbf{p}_i \in \mathbb{R}^d$, to represent a dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$, where $\mathbf{x}_i \in \mathbb{R}^D$. This section discusses how such projections may be visually represented.

Many dimensionality reduction techniques allow an arbitrary target dimension $d \leq D$. For $d > 3$, a d-dimensional projection may be visualized using typical high-dimensional data visualization techniques (discussed in Sec. 2.5), which is recommended by some authors [133]. However, notice that the elements of $\mathbf{p}_i$ are generally difficult to interpret, in contrast to the elements of $\mathbf{x}_i$, which often correspond to meaningful features.

Two-dimensional projections are arguably the most widespread alternative [133]. Such projections are typically represented by scatterplots in Cartesian coordinates. This representation is illustrated by Fig. 2.13, where each point $\mathbf{p}_i$ is also colored according to a pre-defined category y_i assigned to its corresponding observation $\mathbf{x}_i$.

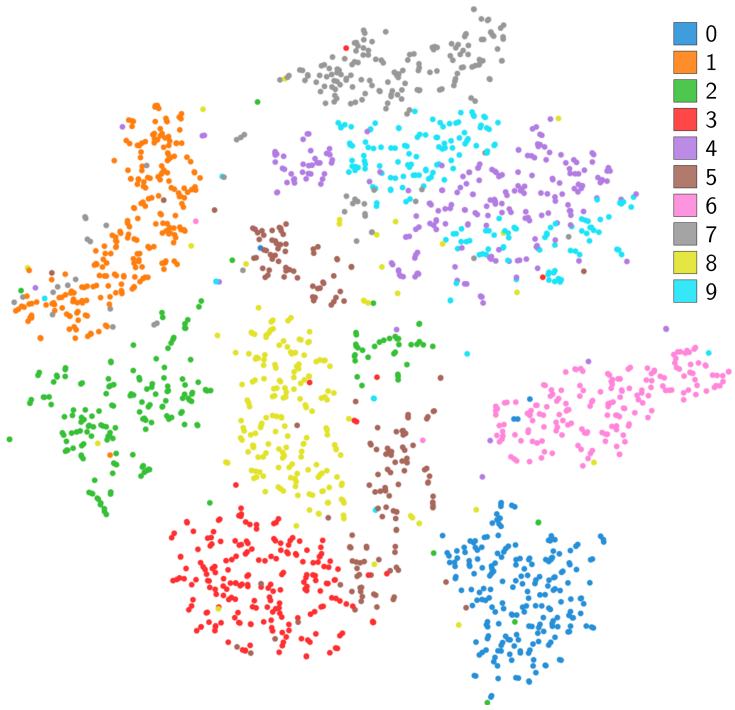


Figure 2.13: Projection represented by a scatterplot.

Notice how Fig. 2.13 allows assessing the relationships between points, presence of (visual) clusters, and overall data distribution. Naturally, because dimensionality reduction techniques generally provide few *quality* guarantees, projections must be interpreted cautiously.

A three-dimensional projection can be represented by an interactive two-dimensional scatterplot where the user chooses the viewpoint (see Fig. 2.14). This alternative has been heavily criticized [133], mostly because some viewpoints may lead to severely misleading representations. This issue has motivated the development of specific visual aids for explaining such projections [27]. However, such visual aids still have interpretation challenges, which justifies our focus on two-dimensional projections.

Instead of coloring each point $\mathbf{p}_i$ according to its category y_i, it is also common to color each point $\mathbf{p}_i$ according to $x_{i,k}$, the value of feature k in observation $\mathbf{x}_i$. As an example, this allows assessing whether a (visual) cluster is uniform with respect to a particular feature. In interactive scatterplots, the user is also commonly able to select (brush) sets of points to inspect the corresponding observations.

Examples of visualizations commonly integrated with projection scatterplots include biplot axes [47, 27], which generally attempt to represent the *direction* of increase in each feature, and axis legends, which attempt to explain the *relationship* between each feature and the Cartesian coordinate axes [27] (see Fig. 2.14).

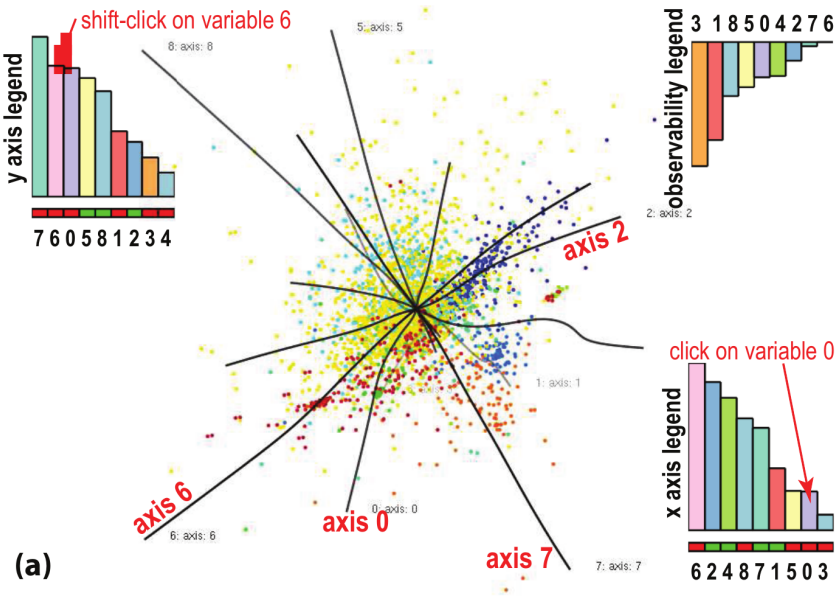


Figure 2.14: Three-dimensional projection represented by an interactive two-dimensional scatterplot, including biplot axes and axis legends. Source: Coimbra *et al.* [27].

Several visualizations are also dedicated to explaining the semantics associated to (visual) *neighborhoods*. For instance, the visualization proposed by Silva *et al.* [134] attempts to represent which features are *most responsible* for the neighborhoods observed in a projection (see Fig. 2.15).

In its simplest definition, clustering is the task of partitioning the observations in a dataset into clusters (sets of observations), such that *similar* observations belong to the same cluster [108]. In some cases, it may also be useful to cluster a projection (rather than a dataset). For instance, Paulovich *et al.* [117] cluster projections of text-document datasets, and represent each (visual) cluster by a word cloud obtained from the corresponding observations, as illustrated by Fig. 2.16. This representation also aims to provide an intuitive overview of the *semantics* associated to each visual cluster.

As we already mentioned, dimensionality reduction techniques generally offer very few guarantees with respect to preserving the data structure in a projection. Therefore, the task of assessing projection *quality* is highly important. If a projection was created using a technique that is based on minimizing a cost function, a possibility is to inspect the resulting cost. However, this offers only a very coarse summary of projection quality, which may be very hard to interpret. Alternatively, several works have proposed metrics that evaluate a projection in a finer level of detail [131, 7, 101, 102].

For instance, Martins *et al.* [101, 102] propose views that highlight missing and false projection neighbors. Figure 2.17 illustrates the missing neighbors view. This view connects a selected point $\mathbf{p}_i$ to its missing neighbors, which are defined as points that should

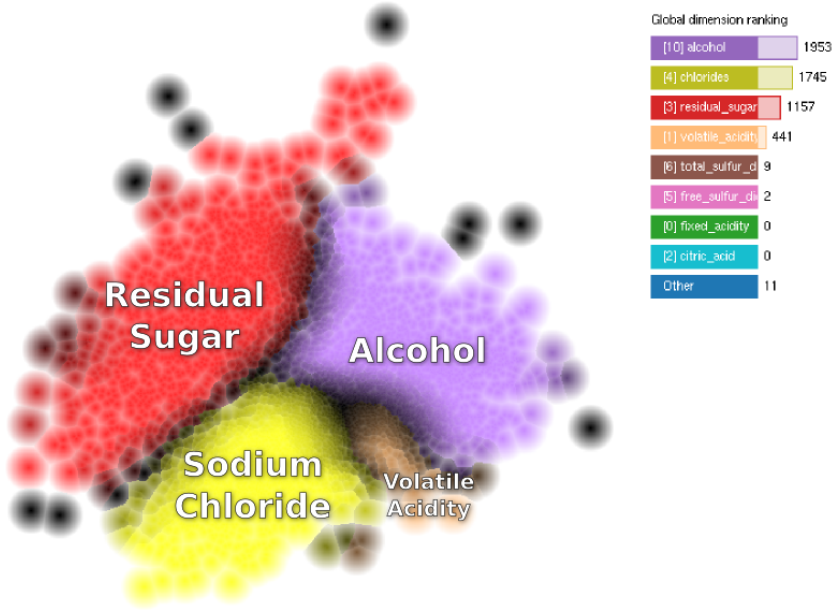


Figure 2.15: Projection represented by a scatterplot. The labels and colors indicate which features are *most responsible* for the *neighborhoods*. Source: Silva *et al.* [134].

Figure 2.16: Projection of a text document dataset. Each visual cluster is replaced by a word cloud. Source: Paulovich *et al.* [117].

be placed closer to $\mathbf{p}_i$ according to some criteria based on the original dataset $\mathcal{D}$.

It is important to emphasize that such projection *error* visualizations address a very different task in comparison to the previous examples, which were mostly concerned with *explaining* projections.

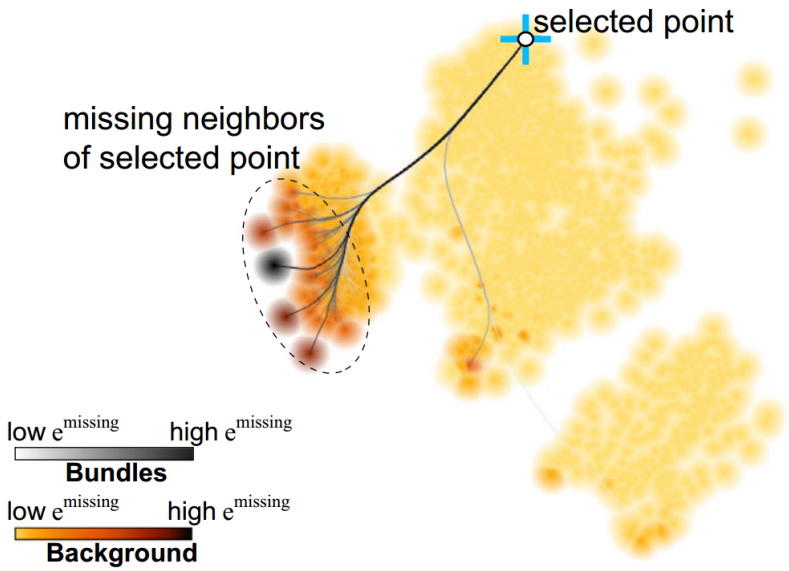


Figure 2.17: Projection represented by a scatterplot. The bundled trails connect a selected point to its missing neighbors. Source: Martins *et al.* [101].

Chapter 3

Interactive image segmentation using superpixels

As already mentioned in Sec. 2.2, user interaction is essential for effective image segmentation. We will focus on purely image-based segmentation, where the user aids the machine mostly in recognition (as opposed to delineation). Recall that we defined recognition as establishing the approximate localization of the objects of interest, and delineation as discovering precisely which pixels belong to each of these objects.

This chapter presents a new interactive segmentation technique based on the image foresting transform (*IFT*, introduced in Sec. 3.1).

Our interest on the image foresting transform for segmentation stems from several advantages that the *IFT algorithm* has over other techniques that find *optimum cuts* in graphs [26]. For instance, some graph cut variants tend to require more *seeds*, since they tend to favor smaller boundaries [26]. Most importantly, IFT-based segmentation methods are capable of segmenting multiple objects in linear or *linearithmic* time [42], while simultaneously segmenting more than two objects using graph cut methods based on the maximum flow algorithm is an NP-hard problem [42, 26].

Our technique extends existing IFT-based segmentation techniques by employing *superpixels* as atomic units (as opposed to pixels). Superpixels are small, cohesive regions within an image, which are expected to belong to a single object of interest [143, 155, 100].

This new technique has two main advantages over its pixel-based counterparts. Firstly, it enables faster graph-based interactive segmentation of very large images. Secondly, it potentially enables extracting better features than those extracted from fixed-size windows around pixels [143, 155, 100]. However, finding appropriate superpixel features for image segmentation has proved a challenging task in preliminary experiments, and partially motivates our interest in feature space exploration (Ch. 4).

Our proposed segmentation technique can be summarized as follows. Firstly, the input image is *oversegmented* into superpixels. Seed pixels defined by the user associate

This chapter is based on the following publication:
Paulo E. Rauber, Alexandre X. Falcão, Thiago V. Spina, and Pedro J. de Rezende. Interactive segmentation by image foresting transform on superpixel graphs. In *Proceedings of the 2013 XXVI Conference on Graphics, Patterns and Images*, SIBGRAPI '13, pages 131–138, Washington, DC, USA, 2013. IEEE Computer Society.

a label to some of these superpixels. A superpixel graph is created to represent the oversegmentation: each superpixel corresponds to a vertex, and edges connect superpixels that are adjacent in the input image. An image foresting transform is then applied to associate a label to each superpixel, exploring the affinity between labeled and unlabeled superpixels.

We have compared our new technique to a pixel-based counterpart, already established in the literature [32], and found it very promising. Our proposal can be enhanced in several ways, which are noted in the appropriate sections, and can be explored by future works.

Another contribution described in this chapter is the development of novel *robot users* to facilitate the evaluation of interactive segmentation methods. The idea of robot users has already been explored in the literature [80], with the main objective of avoiding the costs and biases involved in evaluation by real users. These robots work by creating seeds from the segmentation ground truth, and simulate interactive segmentation by real users. We have developed two robot users that attempt to mimic the behavior of users with expertise on the presented techniques. We also implemented one of the robots described in [80], which uses a strategy that may be more similar to that of non-expert users.

This chapter is organized as follows. Section 3.1 explains the image foresting transform, and is essential to understanding the segmentation techniques described in Section 3.2. Section 3.3 details the robot users, and the results obtained by our technique. Finally, Section 3.4 summarizes our findings.

3.1 Image foresting transform

The image foresting transform (IFT) is a method employed by several techniques based on graph connectivity. These techniques have been successfully applied to solve problems in diverse areas, such as image processing [42], image analysis [42], pattern classification [115], and data clustering [127]. Given a graph and a suitable path cost function, the *IFT algorithm* finds an *optimum-path forest*, which represents the path with the lowest cost ending at each vertex [42]. Section 3.2 describes how graphs may be created from images, while this section details the general IFT algorithm, which can be seen as a generalization of Dijkstras's algorithm [34].

Consider the finite graph $G = (V, E)$, and let $v \in V$ be a vertex. We denote a path π_v ending at v by a sequence of (consecutively adjacent) vertices. We denote by $\pi_u \cdot (u, v)$ the extension of a path π_u ending at $u \in V$ by an edge $(u, v) \in E$. A path $\pi_v = v$ is said to be *trivial*.

Let f be a real-valued path cost function that assigns a *cost* $f(\pi_v)$ to any path π_v in G. For our purposes, a path π_v ending at v is said to be *optimum* if $f(\pi_v) \leq f(\tau_v)$ for any other path τ_v ending at v in G. In other words, a path ending at v is optimum if no other path ending at v has lower cost. Since paths may have completely arbitrary costs, the optimum paths are not necessarily trivial.

If we let $\Pi_v(G)$ denote the set of all paths in G ending at v, the cost $C(v)$ of an

optimum path ending at v is given by

$$C(v) = \min_{\pi_v \in \Pi_v(G)} f(\pi_v). \tag{3.1}$$

As long as the path cost function f is *smooth*, as defined in [42], the IFT algorithm may be used to find a solution to this minimization problem. In such cases, any solution may be represented by a corresponding directed acyclic graph called *optimum-path forest* (OPF). Examples of smooth functions include those used in the next sections, and the typical sum of non-negative edge weights in Dijkstra's algorithm.

The OPF may be represented by a predecessor function P, which assigns each vertex $v \in V$ to its predecessor $P(v) \in V$ in an optimum path. Exceptionally, if $P(v) = \varnothing$, then the trivial path $\pi_v = v$ is optimum, and v is said to be a *root* of the forest.

The IFT algorithm is presented in Alg. 1. The root $R(v)$ associated to each vertex v could be obtained using the predecessor function P. However, it is more efficient to obtain this information during the procedure that finds the OPF. In lines 12 and 18, π_u denotes the (optimum) path from $R(u)$ to u given by the current predecessor function P.

Algorithm 1 General IFT Algorithm

Input: Graph (V, E), path cost function f, empty priority queue Q.
Output: Predecessor function P, optimum cost function C, root function R.

```
 1: for each v ∈ V do
 2:    P(v) ← ∅
 3:    R(v) ← v
 4:    C(v) ← f(v)
 5:    if C(v) ≠ +∞ then
 6:       Insert v into Q
 7:    end if
 8: end for
 9: while Q ≠ ∅ do
10:    Remove u from Q such that C(u) is minimum
11:    for each v such that (u, v) ∈ E and C(u) < C(v) do
12:       if f(π_u · (u, v)) < C(v) then
13:          if C(v) ≠ +∞ then
14:             Remove v from Q
15:          end if
16:          P(v) ← u
17:          R(v) ← R(u)
18:          C(v) ← f(π_u · (u, v))
19:          Insert v into Q.
20:       end if
21:    end for
22: end while
```

Lines 1–8 initialize estimates to consider trivial paths. The vertices with finite costs are inserted into Q, as root candidates. The vertices with optimum trivial paths will become roots of the forest. The main loop (lines 9–22) finds an optimum path from the

eventual roots to each vertex u, in non-decreasing order of cost. At each iteration, a path of minimum cost $C(u)$ is obtained, for some vertex u, and u is removed from the priority queue Q. Importantly, ties in minimum cost in Q are typically broken using a first-in-first-out policy. The remaining lines evaluate whether the path that reaches a vertex v through u has lower cost than the current estimate for optimum path ending at v, and update Q, $C(v)$, $R(v)$ and $P(v)$ accordingly.

We refer to [42] for further considerations about the performance and correctness of the IFT algorithm.

3.2 Segmentation techniques

This section presents our segmentation technique based on superpixels (Sec. 3.2.1), and the technique based on pixels (Sec. 3.2.2) that we used as baseline for comparison. These techniques receive an image and a set of labeled pixels (seeds) for each object of interest, which can be drawn by the users using brushes of different colors (see Fig. 3.4).

We let a d-dimensional image $\boldsymbol{I}$ be a function $\boldsymbol{I} : D_{\boldsymbol{I}} \to \mathbb{R}^c$, where $D_{\boldsymbol{I}} \subseteq \mathbb{Z}^d$ is the image domain, and c is the number of channels. An element $\mathbf{u} \in D_{\boldsymbol{I}}$ is called a pixel, and $\boldsymbol{I}(\mathbf{u}) \in \mathbb{R}^c$ is the value of pixel $\mathbf{u}$. If we let $\boldsymbol{I}(\mathbf{u}) = (I_1(\mathbf{u}), \ldots, I_c(\mathbf{u}))$, then $I_j : D_{\boldsymbol{I}} \to \mathbb{R}$ is called channel j.

In this chapter, we are mostly concerned with two-dimensional ($d = 2$) color images ($c = 3$). We employ the *YCbCr* color space, although any other color space may be equally appropriate. In case $c = 1$, we also denote an image $\boldsymbol{I}$ simply by I.

We represent a segmentation by an image $L : D_{\boldsymbol{I}} \to \{1, \ldots, C\}$ that maps every pixel of $\boldsymbol{I}$ to one of the C objects of interest.

3.2.1 Superpixel-based segmentation

The first step in our method is to generate an oversegmentation of the input image (see Fig. 3.1). Any method may be used for this purpose. We describe here an approach based on the IFT-watershed from grayscale markers [97].

Firstly, we compute a so-called gradient magnitude image $F' : D_{\boldsymbol{I}} \to \mathbb{R}$ that highlights the edges in the input image $\boldsymbol{I} : D_{\boldsymbol{I}} \to \mathbb{R}^3$ (see Fig. 3.2). The image F' is given by

$$F'(\mathbf{u}) = \sum_{\mathbf{v} \in E(\mathbf{u})} \left[\sum_{j=1}^{c} \alpha_j (I_j(\mathbf{u}) - I_j(\mathbf{v}))^2 \right]^{\frac{1}{2}}, \tag{3.2}$$

where $E(\mathbf{u})$ is the set of 8-neighbors of pixel $\mathbf{u}$ in the image domain $D_{\boldsymbol{I}}$. The scalars $\alpha_i \in [0, 1]$ can be used to attribute different weights to each color component. In *YCbCr*, for instance, they may assign less weight to the intensity component Y, making F' more robust to changes in illumination. Based on previous experience with IFT-based segmentation methods, we chose $\alpha_1 = 1/5$ and $\alpha_2 = \alpha_3 = 1$. Next, we obtain an image F by rescaling F' into a pre-defined discrete range of values, which is required for the next step.

The next step is an IFT-watershed from grayscale markers. A classical watershed transform on F can be imagined as a process where the image surface is flooded by *water*

Figure 3.1: Oversegmentation superimposed on input image.

originating at each local minima. The water from each source reaches several level sets until it contacts water from other sources, at locations that correspond to the *ridges* of F. These ridges define boundaries between regions (superpixels).

The sizes of these superpixels may be (indirectly) controlled by building a component tree and removing *basins* with volume below a certain threshold τ [109], resulting in an image $H : D_{\boldsymbol{I}} \to \mathbb{R}$ such that $H(\mathbf{u}) \geq F(\mathbf{u})$ for all $\mathbf{u} \in D_{\boldsymbol{I}}$. The threshold τ is a hyperparameter of our new method.

After the unwanted basins are removed, the IFT-watershed from grayscale markers considers a graph $G = (V, E)$, where $V = D_{\boldsymbol{I}}$ is the set of pixels, and $(\mathbf{u}, \mathbf{v}) \in E$ if and only if $\mathbf{u}$ is a 8-neighbor of $\mathbf{v}$. The path cost function f for this transform is given by

$$f(\mathbf{u}) = \begin{cases} H(\mathbf{u}) & \text{if } \mathbf{u} \in \mathcal{R} \\ H(\mathbf{u}) + 1 & \text{if } \mathbf{u} \notin \mathcal{R} \end{cases}$$
$$f(\pi_{\mathbf{u}} \cdot (\mathbf{u}, \mathbf{v})) = \max\{f(\pi_{\mathbf{u}}), F(\mathbf{v})\}. \tag{3.3}$$

The set $\mathcal{R} = \{\mathbf{u} \notin Q \mid P(\mathbf{u}) = \varnothing\}$ contains the definitive roots of the optimum-path forest, and is updated on-the-fly during the algorithm. This detail enables a single root to *conquer* its entire *plateau* in the image surface [97].

The IFT algorithm results in a root map R that partitions the original domain $D_{\boldsymbol{I}}$ into a set of superpixels (regions) $\mathcal{S}$, such that each superpixel $s_{\mathbf{u}} \in \mathcal{S}$ is given by $s_{\mathbf{u}} = \{\mathbf{v} \in V \mid R(\mathbf{v}) = \mathbf{u}\}$, for some root $\mathbf{u} \in \mathcal{R}$. Intuitively, the root map R maps each pixel to the flooding source that conquered it. If R is seen as an image, its connected components correspond to the superpixels.

Figure 3.2: Gradient magnitude image corresponding to Fig. 3.1.

Regardless of the method employed to oversegment the image into a set of superpixels $\mathcal{S}$, the next step is to associate a feature vector (or observation) $\mathbf{x}_i$ to each superpixel $s_i \in \mathcal{S}$. For simplicity, we consider the mean color of the superpixel s_i as its feature vector $\mathbf{x}_i$, which is given by

$$\mathbf{x}_i = \frac{1}{|s_i|} \sum_{\mathbf{u} \in s_i} \boldsymbol{I}(\mathbf{u}). \tag{3.4}$$

However, superpixels enable extracting features from cohesive regions, which may be better than those extracted from fixed-size windows around pixels [143, 155, 100]. As we already mentioned, finding such features has proved a challenging task in preliminary experiments, and partially motivates the work presented in the next chapters.

The next step in our segmentation method requires another image foresting transform. We will redefine some mathematical objects to keep the notation succinct and consistent with Alg. 1.

In this step, we consider a graph $G = (V, E)$, where $V = \mathcal{S}$ is the set of superpixels, and $(u, v) \in E$ if and only if a pixel in u is 4-neighbor of a pixel in v. Denoting by $L(u)$ the label given by the user to the superpixel u, and letting $L(u) = 0$ when the superpixel u is not labeled, the connectivity function f is given by

$$f(u) = \begin{cases} 0 & \text{if } L(u) \neq 0 \\ +\infty & \text{if } L(u) = 0 \end{cases}$$

$$f(\pi_u \cdot (u, v)) = \max\{f(\pi_u), w(u, v)\}, \tag{3.5}$$

where $w(s_i, s_j)$ is the (weighted) Euclidean distance between the corresponding superpixel feature vectors $\mathbf{x}_i$ and $\mathbf{x}_j$. We employ the same weights α_k used in Equation 3.2.

Intuitively, the path cost function f makes large differences between adjacent superpixels act as barriers in paths that go through them.

The desired segmentation can be obtained from the root map R that results from the IFT algorithm. Each unlabeled superpixel s is associated to the label $L(R(s))$ associated to its root $R(s)$, which is always labeled. See Figure 3.3 for an illustrative example.

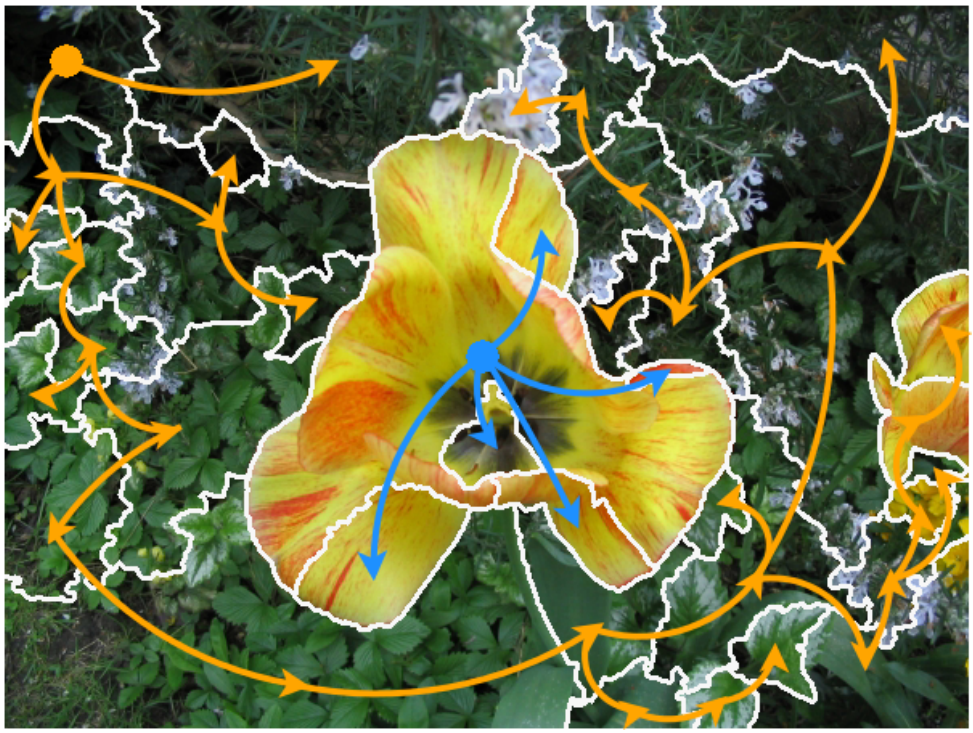


Figure 3.3: Illustrative optimum-path forest with two roots (top left and center). Superpixels are larger than usual.

Note that a user could potentially label a single superpixel with multiple labels using brushes. However, this should be infrequent, since the oversegmentation should respect the borders of the correct segmentation. We do not address this issue in our experiments, and arbitrarily choose one of the labels associated to each superpixel. This could be addressed by applying an oversegmentation on a finer scale in ambiguous superpixels.

Our method allows multi-object segmentation in time $O(|V| \log |V|)$ [42], and this time complexity could be further improved by discretizing the edge weights given by the function w. This is a remarkable advantage in comparison to graph-cut methods that are based on the maximum-flow algorithm [26]. Figures 3.4 and 3.5 present multi-object segmentation results based on our method.

Further efficiency gains can be obtained by a differential IFT, which is capable of adapting an optimum-path forest to label changes between iterations [41].

Although our presentation focuses on two-dimensional image segmentation, we have also implemented volumetric segmentation based on supervoxels (see Fig. 3.6).

Figure 3.4: Seeds drawn by a user for multi-object segmentation.

3.2.2 Pixel-based segmentation

The pixel-based segmentation method is very similar to the method based on superpixels, and may be seen as its counterpart. Given an image $\boldsymbol{I} : D_{\boldsymbol{I}} \to \mathbb{R}^3$, this method is based on a graph $G = (V, E)$, where $V = D_{\boldsymbol{I}}$ is the set of pixels, and $(\mathbf{u}, \mathbf{v}) \in E$ if and only if pixel $\mathbf{u}$ is a 4-neighbor of pixel $\mathbf{v}$.

The path cost function f for the corresponding image foresting transform is analogous to the one in the previous section. The feature vector $\mathbf{x}$ for pixel $\mathbf{u}$ is simply given by its color $\mathbf{x} = \boldsymbol{I}(\mathbf{u})$.

It is important to note that the graph G has considerably more vertices than its superpixel counterpart. This is a major disadvantage with respect to time complexity. Although creating the superpixel graph also requires time, this step can be conducted before the interactive segmentation procedure begins.

Both pixel and superpixel techniques can be enhanced by the combination of supervised and unsupervised learning described in [32].

3.3 Experiments

This section describes the experiments conducted to evaluate our new technique. Seeking to reduce costs and biases associated with evaluation by real users, we have developed robot users that, given the segmentation ground truth, attempt to simulate expert and non-expert users. These robots are described in Section 3.3.1, and the experimental results are described in Section 3.3.2.

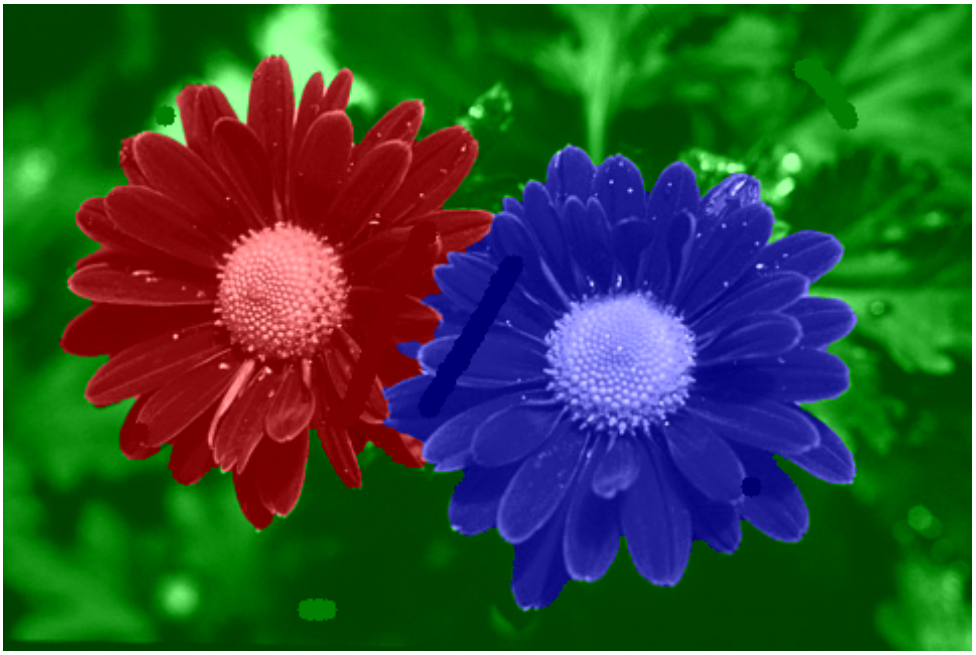


Figure 3.5: Resulting superpixel-based multi-object segmentation. Different hues represent different segments.

3.3.1 Robot Users

We have implemented three robot users that employ different strategies to label pixels. Two of them are original contributions. A comparison between the behavior and efficacy of our robots and real users could be explored in future works.

Although both segmentation techniques that we consider can be used for multi-object segmentation, we have compared them in the context of binary segmentation (labeling pixels as either object of interest or background), since this is the scenario most often used to evaluate segmentation techniques. In these evaluations, the segmentation is compared to a ground-truth image, and a measure is used to quantify the efficacy of the particular segmentation technique.

Each robot user creates an ordered list $\mathcal{P}$ of seed candidates (pixels). The order defines the priority for labeling a given candidate. This list can be used in several ways, one of which is described in Section 3.3.2.

Geodesic robot

The geodesic robot was introduced by [80], and is the simplest that we consider (see Fig. 3.7). We employ this robot as an attempt to mimic the behavior of an untrained user trying to segment an image employing very little effort. Differently from [80], we do not start from pixels labeled by real users, making our method completely automatic.

In the first iteration of interactive segmentation, this robot creates a list $\mathcal{P}$ that contains the geodesic center of every region of interest (in our case, object and background), computed from the ground truth.

In the next iterations, this robot computes an image E, such that $E(\mathbf{u}) = 0$ if pixel $\mathbf{u}$

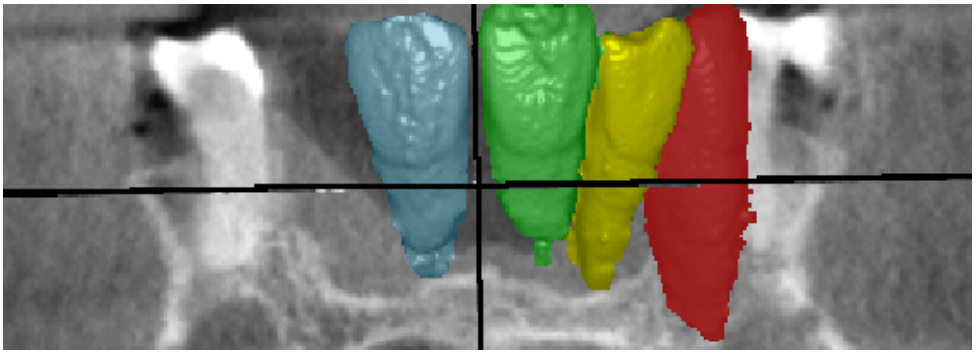


Figure 3.6: Rendering of supervoxel-based volumetric segmentation of teeth images. Different hues represent different segments.

was correctly labeled by the previous segmentation, and $E(\mathbf{u}) = \lambda$ if the correct label for $\mathbf{u}$ is λ. The goal is to create a list $\mathcal{P}$ containing the geodesic center of every (incorrectly labeled) connected component in the image E in decreasing order of minimum distance between such center and the border of its component. In other words, larger error regions should have higher priority for labeling.

The robot accomplishes this last step by using E to find every connected component with the same label through a breadth-first search. Using the Euclidean distance transform (EDT), which can be efficiently implemented by the IFT [42], the robot finds the geodesic center for every component and sorts them into the list $\mathcal{P}$.

Superpixel robot

The *superpixel* robot attempts to mimic an expert in superpixel-based segmentation (see Fig. 3.8).

Firstly, the robot oversegments the input image into superpixels. The robot then finds superpixels on the borders of the ground truth (*i.e.*, adjacent to superpixels with a different correct label). These superpixels are sorted in increasing order of minimum distance between their feature vectors and the feature vector of some adjacent superpixel with a different label. The idea behind this heuristic is that larger similarity between adjacent superpixels with different labels indicates a higher risk of mislabeling by our segmentation technique based on superpixels. Finally, the robot creates the ordered list $\mathcal{P}$ containing the geodesic centers of these superpixels.

Pixel robot

The *pixel* robot attempts to mimic an expert in pixel-based segmentation (see Fig. 3.9). Its objective is to create seed pixels near pixels on a ground truth border that have low gradient magnitude considering the original image (*weak edges*). This heuristic is based on the idea that labeling these places avoids *leakages*.

The robot begins by computing a gradient magnitude image F from an original image $\boldsymbol{I} : D_{\boldsymbol{I}} \to \mathbb{R}^3$, as in Section 3.2.1. Then, a binary image $B : D_{\boldsymbol{I}} \to \{0, 1\}$ is created, such that $B(\mathbf{u}) = 1$ if and only if $||\mathbf{u} - \mathbf{v}|| \leq \alpha$, for a pixel $\mathbf{u} \in D_{\boldsymbol{I}}$, a pixel $\mathbf{v} \in D_{\boldsymbol{I}}$ on

Figure 3.7: Discs centered on seed pixels generated by the geodesic robot. At each iteration, indicated by the numbers, the error components are found, and a number (up to a fixed limit) of seeds is chosen in their geodesic centers.

a ground truth border, and a parameter α. Intuitively, every pixel located on a ground truth border is the center of a disc of radius α in B.

For every pixel $\mathbf{u}$ such that $B(\mathbf{u}) = 1$, and $\mathbf{u}$ has some neighbor $\mathbf{v}$ such that $B(\mathbf{v}) = 0$, the robot associates the gradient magnitude $F(\mathbf{w})$, where $\mathbf{w}$ is the nearest pixel to $\mathbf{u}$ that is on a ground truth border. Intuitively, each pixel on a border of B is associated to the gradient magnitude of its nearest pixel on the border of the ground truth. This is analogous to eroding/dilating the ground truth and associating the gradient magnitude of the original pixels on the ground truth border to the corresponding eroded/dilated pixels.

The robot then creates a list $\mathcal{P}$ of pixels on the border of B, sorted in increasing order of gradients from their associated pixels on the ground truth border.

3.3.2 Results

We have chosen three widespread datasets to compare the techniques that we presented: *grabcut* [128, 16], *geodesic star* [59] and *Weizmann* (single object, first human subject) [4]. These datasets contain, respectively, 50, 151, and 100 photographies and ground truths.

The experiments were conducted as follows. For a total of T iterations, given the list $\mathcal{P}$ created by a robot and a parameter n, we chose the first $n/2$ seeds inside an error component from the object and $n/2$ seeds inside an error component from the background. We correctly label a disk (*marker*) of a given radius around these seeds, which are used in the next iteration of interactive segmentation. Note that the seed candidates are generated only once by the pixel and superpixel robots, while the geodesic robot needs to compute the geodesic centers of the error components at each iteration.

Figure 3.8: Discs centered on seed pixels generated by the superpixel robot. Similar superpixels with different labels have higher priority for labeling. The numbers indicate iterations.

Moreover, we consider some constraining details when placing markers: if the marker is too near a ground truth border (as defined by a parameter), its size is reduced. This is done since the ground truth images are often imperfect, and including markers too near the ground truth border could create *artificial* leakages that would not be created by real users. We also chose not to draw a marker centered on seed pixels that are already labeled, since that would disproportionately reduce the total area labeled by the pixel robot.

As in [4], we have chosen the *f-score* as a measure of efficacy. For a given image, the *precision* can be understood as the proportion of pixels *labeled as belonging to the object* (as opposed to the background) that were correctly labeled, while the *recall* can be understood as the proportion of *object pixels* that were correctly labeled. The f-score is the harmonic mean between these two quantities, which combines them into a single scalar that balances the different types of errors. Naturally, the f-score lies in the interval $[0, 1]$ and higher values are desirable.

We have chosen the following parameters for our experiments: for each technique, using each robot, we ran $T = 8$ iterations, choosing up to $n = 8$ pixels at each iteration. The radius of the discs centered on the seeds was 5 pixels, and could be reduced to 1 whenever the seed was too near a ground-truth border (2 pixels from a ground truth border was considered a *safe* distance). The superpixel robot had a volume threshold of $\tau = 15000$. For the pixel robot, we chose $\alpha = 15$ (erosion/dilation radius). We removed a total of five images from the datasets, since they had objects of interest too small for the pixel robot. Figures 3.7, 3.8 and 3.9 illustrate some markers created by the robots (for the superpixel segmentation method) until the fifth iteration. Note that, for that

Figure 3.9: Discs centered on seed pixels generated by the pixel robot. Each seed is associated to the *gradient magnitude* of its nearest pixel in the ground truth border. Lower gradient magnitude indicates higher priority for labeling. The numbers indicate iterations.

particular image, markers created by the pixel and superpixel robots were chosen only up to the second iteration, at which point there were no longer seed candidates inside error components.

The volume threshold for our superpixel segmentation technique, which controls superpixel size, was chosen as $\tau = 150$. However, different image categories may benefit from different choices.

The small number of seeds per iteration allows us to see the quality of the generalizations made by each method. The seeds created by the superpixel and pixel robots are purposefully far from the ground truth border to highlight differences in delineation. The parameters were also chosen to enable for good convergence and to create interactions (subjectively) similar to what would be expected from real users. As noted earlier, empirically establishing these parameters to match real users could be explored by future works.

Our results are summarized in Figures 3.10, 3.11 and 3.12. The graphs display the mean f-score obtained by the two techniques for every combination of dataset and robot.

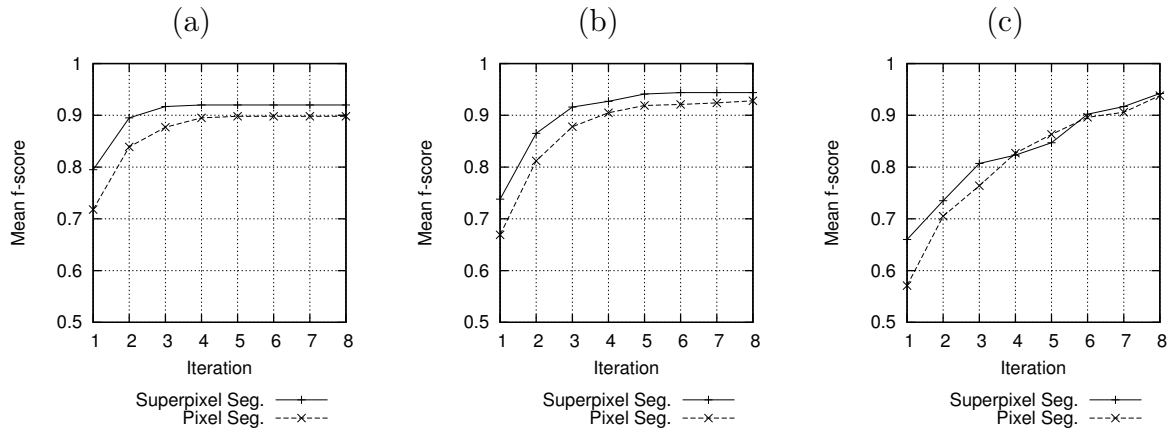


Figure 3.10: Mean f-score for images in the Weizmann dataset (a) Superpixel Robot (b) Pixel Robot (c) Geodesic Robot

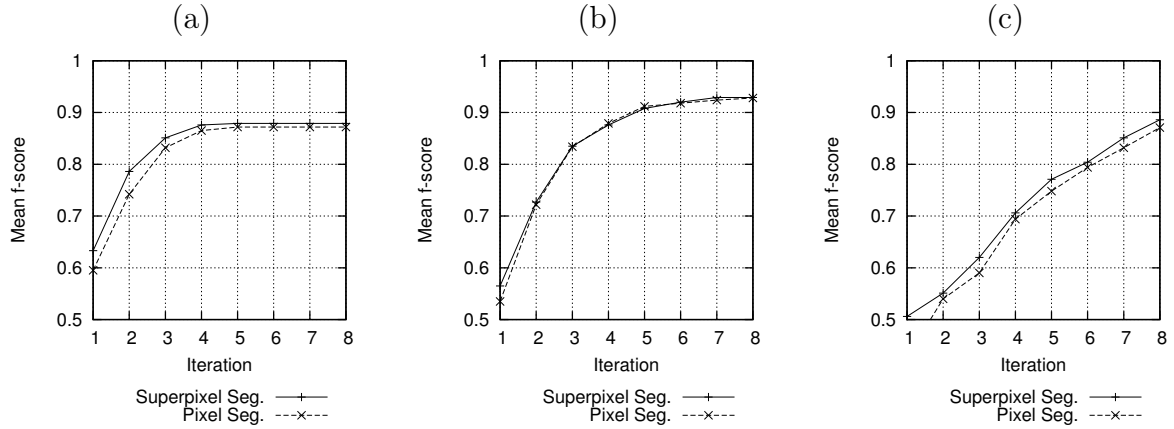


Figure 3.11: Mean f-score for images in the geostar dataset (a) Superpixel Robot (b) Pixel Robot (c) Geodesic Robot

Our superpixel-based technique achieved higher efficacy (with respect to mean f-score) in the majority of cases, most notably on the Weizmann dataset, and in the first iterations. In the few cases where the mean f-score was lower at some iteration, its results were not considerably inferior. Although these efficacy results are certainly positive, we only intend to claim that our superpixel-based technique is promising. For the sake of perspective, Figures 3.13 and 3.14 illustrate a difference of 9.5% in f-score.

We also timed a single (manually seeded) iteration of interactive segmentation considering a reasonably large image composed of 4096×4096 pixels, using the same parameters employed in the previous experiments (which consider smaller images). On a typical desktop computer (*Intel i7-2600* at 3.4 GHz), the superpixel-based technique requires approximately 16417ms (milliseconds) for setup (including volume filtering, IFT-watershed from grayscale markers, and superpixel graph creation), but only approximately 128ms per iteration (averaged over three runs). The pixel-based technique requires only 817ms for setup, but 7958ms per iteration (approx. 62 times the time required by its superpixel counter-

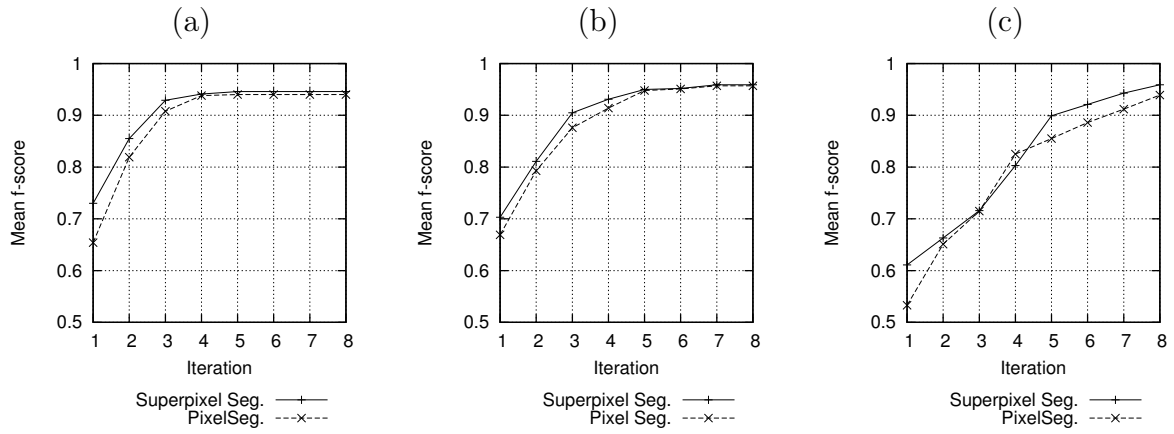


Figure 3.12: Mean f-score for images in the grabcut dataset (a) Superpixel Robot (b) Pixel Robot (c) Geodesic Robot

part). This makes our superpixel-based technique particularly attractive for interactive segmentation of large images, since its setup may be conducted before user involvement, and generally several iterations are required to achieve satisfactory results. Reducing the number of superpixels would increase even more the advantage of our method.

3.4 Conclusion

In this chapter, we presented a new interactive segmentation technique based on superpixels, and compared it to its pixel-based counterpart. The experimental evaluation shows that our new technique is promising, particularly due to its advantages on efficiency and potential for use in conjunction with more powerful feature descriptors.

We have also presented novel robot users that can be employed to evaluate interactive segmentation methods. Empirically establishing the similarities between these robots and real users could be explored by future works.

As future works, we also suggest comparing our new technique with other established techniques, the study of superpixel descriptors, multiscale oversegmentation, and superpixel-based differential IFT [41].

Figure 3.13: Superpixel-based segmentation, f-score: 0.985.

Figure 3.14: Pixel-based segmentation, f-score: 0.89. The seeds are the same as the ones used in Figure 3.13.

Chapter 4

Interactive feature selection assisted by projections

This chapter focuses on the task of pattern classification, which we introduced in Sec. 2.3. Recall that we divided pattern classification into two subtasks: representation and learning.

The representation task consists on representing objects of interest as observations (high-dimensional real vectors). In this context, selecting features that are valuable for generalization is a very important problem. For instance, consider image classification. Using too few features can lead to poor generalization; while using too many features can be prohibitively expensive to compute, or even introduce confounding information into the training data [60, 93]. Although deep neural networks recently became able to bypass feature design by dealing directly with raw images, these models pose their own challenges, as we will discuss in Chapter 5. As we have mentioned, feature selection is also challenging in image analysis tasks that are typically considered outside the scope of machine learning, such as image segmentation.

The learning task consists on selecting and fine-tuning learning algorithms once the observations are available. As we have seen in Sec. 2.3, no single algorithm is the best for every situation. Practitioners usually compare algorithms and hyperparameter choices using cross-validation [108]. However, this approach is bounded by the limited feedback that numerical (classification) efficacy measures can provide. As a consequence, when suboptimal results are obtained, designers are often left unaware of which aspects limit classification system efficacy, and what can be done to improve these systems. This and other issues have been referred to as the "black art" of machine learning [35], and motivate our interest in interactive techniques for classification system design.

In the context of high-dimensional data visualization, dimensionality reduction is an important class of highly scalable techniques, which we introduced in Section 2.6. These techniques find a projection that attempts to preserve the so-called structure of a high-dimensional dataset. This structure is characterized by distances between observations, presence of clusters, and overall spatial data distribution [95, 88]. For the purposes of visu-

This chapter is based on the following publication:
Paulo E. Rauber, Alexandre X. Falcão, and Alexandru C Telea. Projections as visual aids for classification system design, 2016. Submitted to Information Visualization (IVI).

alization, dimensionality reduction techniques typically reduce the number of dimensions to two or three. The resulting projections can be visually represented by scatterplots, and enable insight into the structure of the original data [148].

Many types of data have been explored using projections, such as text documents [116], multimedia collections [75], gene expressions [24], and networks [25]. However, projections are rarely used for the task of classification system design. In this context, we propose an interactive visual analytics approach based on dimensionality reduction that supports two (highly interrelated) tasks: predicting classification system efficacy, and improving classification systems through feature space exploration.

This chapter is organized as follows. Section 4.1 reviews our notation and definitions. Section 4.2 places our effort in the contexts of information visualization and machine learning. Section 4.3 summarizes and compares our approach to related works. Section 4.4 details our first contribution: showing how projections can be used as insightful predictors of classification system efficacy. Section 4.5 details our second contribution: showing how the visual feedback given by projections can be integrated into an interactive and iterative workflow for improving system efficacy through qualitative and quantitative data exploration. Section 4.6 provides a critical analysis of the experiments, limitations, and weaknesses of our approach. Importantly, it outlines cases where projections are known to fail as predictors of classification system efficacy, and why such cases do not contradict our proposal. Finally, Section 4.7 summarizes our findings and suggests directions for future work.

4.1 Preliminaries

The following is a concise review of definitions introduced in Ch. 2.

A (supervised) dataset $\mathcal{D}$ is a sequence $\mathcal{D} = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$. Every pair $(\mathbf{x}_i, y_i) \in \mathcal{D}$ is composed of an *observation* $\mathbf{x}_i \in \mathbb{R}^D$, and a *class label* $y_i \in \{1, \ldots, C\}$, where C is the number of *classes*. The j-th element of $\mathbf{x}_i$ corresponds to *feature* j, and is typically measured from an object of interest.

We denote the set of all features under consideration by $\mathcal{F} = \{1, \ldots, D\}$. For any $\mathcal{F}' \subseteq \mathcal{F}$, having $D' \leq D$ features, we denote by $\mathcal{D}_{\mathcal{F}'}$ the dataset corresponding to $\mathcal{D}$ with features restricted to $\mathcal{F}'$.

A *learning algorithm* finds a function, called *classifier*, that maps observations to classes based on generalization from a training (data)set $\mathcal{D}$. *Generalization* is usually evaluated by *cross-validation*, which consists on partitioning the available data into a set for model learning and a set for model evaluation. *Feature selection* aims at finding a small feature subset $\mathcal{F}' \subseteq \mathcal{F}$ such that the restricted training set $\mathcal{D}_{\mathcal{F}'}$ is sufficient for generalization.

Dimensionality reduction finds a *projection* $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$, where $\mathbf{p}_i \in \mathbb{R}^d$, that attempts to preserve the *structure* of an original (unsupervised) dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$, considering that each observation $\mathbf{x}_i$ corresponds to point $\mathbf{p}_i$. For the purposes of visualization, d is usually 2 or 3.

4.2 Related work

Our focus on high-dimensional data visualization based on dimensionality reduction is justified by the scalability of projections with respect to the number of observations and features, which we already discussed in Sections 2.5 and 2.6.

Several visualization techniques have been proposed to help the interactive exploration of projections. Most notably, Tatu *et al.* [141] propose a process for finding *interesting* subsets of features, and displaying the results of dimensionality reduction restricted to these features, with the goal of aiding qualitative exploration. Yuan *et al.* [158] present an interactive tool to visualize projections of observations restricted to selected subsets of features. Additionally, in their tool, features are placed in a scatterplot based on pairwise similarities. This is analogous to the representation we propose in Section 4.5. However, clear differences exist: Yuan *et al.* [158] aim at subspace cluster exploration, while our goal is to provide support for classification system design. This difference is manifested by our additional mechanisms, which include feedback from automatic feature scoring techniques and classification results. The work of Turkay *et al.* [144] also combines scatterplots of observations and features for high-dimensional data exploration, and is also concerned with tasks that are unrelated to classification system design.

Pattern classification is one of the most widely studied problems in machine learning. Since the objective of pattern classifiers is to generalize from previous experience, hyper-parameter search and efficacy estimation are usually performed using cross-validation, as we already discussed in Sec. 2.3. Diagnosing the cause of poor generalization in classification systems is very difficult. Options include using cross-validation to compute efficacy indicators (*e.g.*, accuracy, precision and recall, area under the ROC curve), and learning curves, which show generalization performance for an increasing training set. In multi-class problems, confusion matrices can also be used to diagnose mistakes between classes [45].

In this context, Talbot *et al.* propose the visual comparison of confusion matrices to help users understand the relative merits of various classifiers, with the goal of combining them into better ensemble classifiers. In contrast to their work, we offer finer-grained insight into a single classification system by using projections as a visualization technique. Other visualization systems also aim at integrating human knowledge into the classification system design process. *Decision trees* are particularly suitable for this goal, as they are one of the few easily interpretable classification models [145]. In contrast, Schulz *et al.* [132] propose a framework that can be used to visualize (in a projection) the decision boundary of a support vector machine, a model which is usually hard to interpret. Other works also propose visualizations that consider classification systems as *black-boxes*. They usually provide an interface to study the behavior of such systems under different combinations of data and parameterizations. In this context, Paiva *et al.* [114] present a visualization methodology that supports tasks related to classification based on similarity trees. Similarly to projections, similarity trees are a high-dimensional data visualization technique that maps observations to points in a 2D space, and connects them by edges to represent similarity relationships. In contrast to our methodology for system improvement, their methodology focuses on visualization of classification results and observation

labeling. Finally, the use of visualization techniques to "open the black box" of general algorithm design, including (but not limited to) classification systems, is advocated by Mühlbacher *et al.* [107].

Active learning refers to a process where the learning algorithm iteratively suggests informative observations for labeling. The objective of this process is to minimize the effort in labeling a dataset. Because this is an iterative and interactive process, visualization systems have been proposed to aid in the task, and sometimes include a representation of the data based on projections [65, 68]. However, in these examples, projections do not have a role in improving classification system efficacy.

Feature selection is another widely researched problem in machine learning, because the success of supervised learning is highly dependent on the predictive power of features, as we discussed in Sec. 2.4. The work of Krause et al. [82] is an example of visualization system that aids feature selection tasks by displaying aggregated feature relevance information, which is computed based on feature selection algorithms and classifiers.

4.3 Proposed approach

Our visualization approach aims to support two tasks (**T1** and **T2**), which we introduce in the following sections.

4.3.1 T1: predicting system efficacy

Consider the works presented in Sec. 4.2 that use projections to represent observations in classification tasks (*e.g.*, [65, 68]), or the projections of traditional pattern classification datasets (*e.g.*, [148]). If a projection shows good visual separation between the classes in the training data, and if this is expected to generalize to test data, it is natural to suppose that building a good classifier will be easier than when such separation is absent.

However, there is little evidence in the literature to defend the use of projections as predictors of classification system efficacy. As a consequence, it is unclear *whether* and, even more importantly, *how* insights given by projections complement existing methods of prognosticating and diagnosing issues in the classification pipeline. In Section 4.4, we present a study that focuses precisely on these questions. It is important to emphasize the term *predictor*: we aim at obtaining insights on the ease of building a good classification system by using projections *before* actually building the entire system.

We are aware of a single previous work that studies how projections relate to classifier efficacy [20], which provides evidence that projections showing *well-separated* classes (as measured by the so-called silhouette coefficient) correlate with higher classification accuracies. However, that study has significant limitations. Firstly, characterizing a projection by a single numerical value (the silhouette coefficient) is coarse and uninsightful. To support understanding *how* a classification system relates to what a projection shows on a finer scale, we perform and present our analyses at the *observation* level. Secondly, the silhouette coefficient used in [20] can be severely misleading, since it may be poor (low) even when good visual separation between classes exists. This happens, for instance, when the same class is spread over several compact groups in a projection. Thirdly, we present

a way to improve classification system efficacy (**T2**), whereas [20] only conjectures this possibility.

Consider simple alternatives to visualize classification system issues, such as confusion matrices [45], or listing misclassified observations together with their k-nearest neighbors. While simple to use, these mechanisms have significant limitations: confusion matrices become hard to inspect for a moderate number of classes, while listing does not scale well to hundreds (or even tens) of observations. Most importantly, these alternatives do not encode spatial information about observations in *confusion zones*, which we define in Sec. 4.4.

4.3.2 T2: improving system efficacy

In Section 4.5, we propose a projection-based methodology for interactive feature space exploration that allows selecting features to improve the efficacy of a classification system (**T2**). This methodology is highly dependent on the use of projections as predictors of classification system efficacy (**T1**).

We implement this methodology in a visual analytics tool that links views of projections, representations of feature relationships, feature scoring, and classifier evaluation, in an attempt to provide a cost-effective and easy-to-use way to select features for arbitrary ("black-box") learning algorithms. The visual analytics workflow supported by our system, detailed in Sec. 4.5, is different from those enabled by previously mentioned feature-space exploration systems [158, 144, 126].

4.3.3 Visual analytics workflow

Figure 4.1 illustrates how our approaches towards addressing **T1** and **T2** interact in a (simplified) visual analytics workflow that supports the overall goal of building better classification systems. The entire process can be summarized by a 10-step flowchart. We start our workflow by considering a set of objects of interest (images, for instance). Next, we extract a number of features from these images, transforming them into observations (1). These observations are projected into a 2D view (2). To asses whether we can trust the projection, we may evaluate the various projection error metrics proposed in [101, 102]. If the errors are too high, we repeat step 2 using a different dimensionality reduction technique or parameter settings. Upon obtaining a good quality projection, we assess the visual separation between the classes using our proposed visual tools. If the separation is poor (4), we use our iterative feature exploration/selection tools (**T2**) to prune the feature set under consideration (5). If visual separation is satisfactory (3), we proceed in building and testing a classifier, using the traditional machine learning protocol (6). If testing yields good performance (7), the pipeline ends with a good classification system that can be used in production. If testing reveals poor performance (8), we apply again the visual exploration tool to study what has gone wrong (9), and proceed to designing new features, repeating the process from the beginning (10).

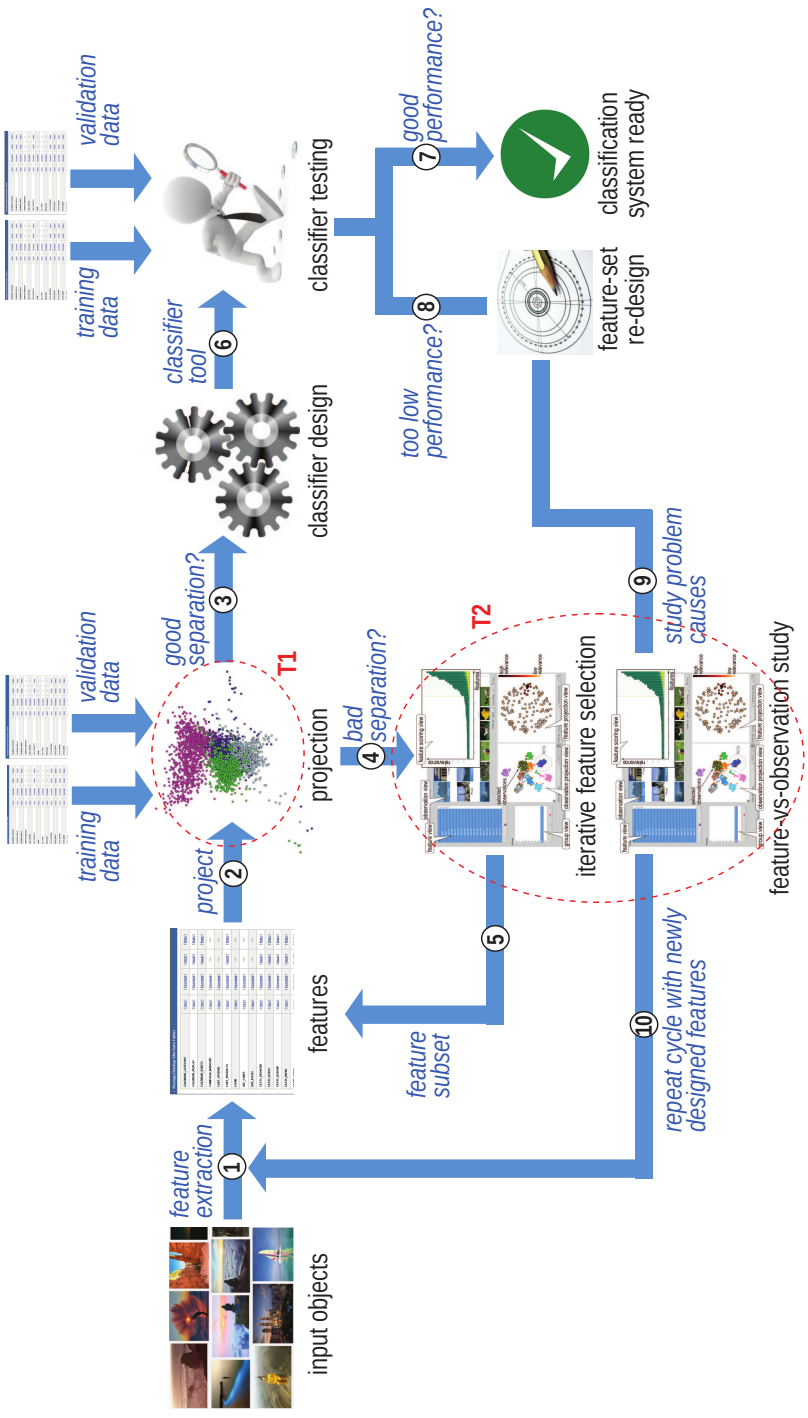


Figure 4.1: Classification system design visual-analytics workflow proposed in this chapter (see Sec. 4.3).

4.4 T1: Predicting system efficacy

As outlined in Sec. 4.3, we start by studying how projections can be used to predict classification system efficacy (**T1**). For this purpose, we conducted experiments on several datasets, which are presented in Secs. 4.4.2 - 4.4.5. Section 4.4.1 details the aspects of the experimental protocol that hold for every dataset under consideration.

4.4.1 Experimental protocol

The first step in our protocol is to randomly partition a dataset into training and test sets (one third of the observations). Following good practice in machine learning, the partitioning is stratified [50], *i.e.*, the ratio of observations belonging to each class is preserved in the test set.

Projections can be created independently for the training and for the test data. These projections can be represented by scatterplots, where each point is colored according to its class label. When displaying classification results for a test set in a scatterplot, we will use triangular glyphs to represent misclassified observations, colored based on their (incorrect) classifications, and rendered slightly darker (for emphasis).

In addition to showing these scatterplots, we also display a metric called *neighborhood hit* (NH) [116]. For a given number of neighbors k (in our experiments, $k = 6$), the NH for a point $\mathbf{p}_i \in \mathcal{P}$ is defined as the ratio of its k-nearest neighbors (except $\mathbf{p}_i$ itself) that belong to the same class as the corresponding observation $\mathbf{x}_i$. The NH for a projection is defined as the average NH over all its points. Intuitively, a high NH corresponds to a projection where the real classes (ground truth) are visually well separated. Therefore, the NH metric is a good quantitative characterization of a projection for our purposes.

The DR technique that we employ in this work is a fast approximate implementation of t-distributed stochastic neighbor embedding (t-SNE, described in Sec. 2.6.4), using default parameters. We chose t-SNE due to its widespread popularity, and demonstrated capacity to preserve neighborhoods in projections [148]. However, our proposal does not depend on this particular technique, and other DR techniques can be used with no additional burden. For instance, we employed LSP [116] in our early work, but decided in favor of t-SNE due to its ability to preserve clusters in projections.

Our workflow requires a projection that preserves well *neighborhoods* from $\mathbb{R}^D$ in $\mathbb{R}^2$. This can be assessed through the projection quality metrics described in [101, 102]. If a projection shows poor quality, it should be discarded (Fig. 4.1, step 2) and not used further in the workflow. Instead, the measures outlined in [101, 102] should be used to improve projection quality. Conversely, if a projection shows good quality, it becomes an excellent candidate for assessing the visual separation between groups, an can be used further in the workflow (steps 3 and 4).

Feature selection will be performed in many of our experiments. We will select a subset of features $\mathcal{F}' \subseteq \mathcal{F}$ to investigate the effect of restricting the input of the DR technique to these features – that is, we will compare the projections of both $\mathcal{D}$ and $\mathcal{D}_{\mathcal{F}'}$. We perform feature selection/scoring using extremely randomized trees [51], using the method described in Sec. 2.4.4 with 1000 trees in the ensemble. Scores are assigned to

features based on their power to discriminate between two given sets of observations. As will become clear in the next sections, the choice of feature selection technique does not affect our proposal. Feature selection is always performed considering only the training set, as this allows assessing the generalization of the selection to the test set.

Learning algorithms will be used to evaluate whether good projections (with respect to perceived class separation) correspond to good classification systems. We consider three distinct algorithms: k-nearest neighbors (KNN, Sec. 2.3.1, using Euclidean distances), soft-margin support vector machines (SVM, Sec. 2.3.3, using radial basis function kernel) [18] and random forest classifiers (RFC, Sec. 2.3.4) [23]. These techniques were chosen for being both widely used in machine learning and representative of distinct classes of algorithms. Note that *any* other classification technique can be used together with our approach, since the techniques are treated as black-boxes, i.e., we assume no knowledge of their inner workings.

Hyperparameter search is conducted by grid search on a subset of the hyperparameter space for each learning algorithm. Concretely, we choose the hyperparameters with highest average accuracy on 5-fold cross-validation on the training set. For KNNs, the hyperparameter is the number of neighbors k (from 1 to 21, in steps of 2). For SVMs, the hyperparameters are the penalty C and the kernel parameter γ (both from 10^{-10} to 10^{10}, in multiplicative steps of 10). For RFCs, the hyperparameters are the number of trees (10 to 500, in steps of 50) and maximum tree depth (from 1 to 21, in steps of 5). In the next sections, we use the term *classifier* to refer exclusively to a particular combination of learning algorithm and hyperparameters trained on the entire training set. The hyperparameters are always found by the procedure outlined in the previous paragraph. In summary, following good machine learning practice, the test set does not affect the choice of hyperparameters.

Classification results are always quantified, in this chapter, by the *accuracy* (AC, ratio between correct classifications and total classifications) on the test set.

Presentation of experiments is uniform across datasets. For each experiment, a high-level claim is first stated. This claim is followed by supportive images, showing projections and classification results. In several cases, some aspect of the problem is altered (*e.g.*, features or observations under consideration), and we show how our projections reflect the expected outcome.

Limitations of our study are discussed in Section 4.6.

4.4.2 Madelon dataset

Data: *Madelon* is a synthetic dataset created by Guyon et al. [61], which contains 500 features and 2 class labels. We split the Madelon training set into training (1332 observations) and test (668 observations) sets, following our experimental protocol. The number of observations in each class is balanced. This artificial dataset was created specifically for the NIPS 2003 feature selection challenge. Only 20 of the 500 features are informative, *i.e.*, useful for predicting the class label. According to its authors, this dataset was designed to evaluate feature selection techniques when features are informative only when considered in groups [61].

Goal 1: Our first goal is to show that, for this dataset, poor separation between classes in the projection corresponds to poor classification accuracy. While this correspondence may appear obvious, it is easy to show that it does not always hold (see Sec. 4.6). Therefore, analyzing the link between visual separation and classification accuracy is worthwhile.

Consider the projection of the training data shown in Fig. 4.2a. The two class labels, represented by distinct colors, are not visually separated in the projection, as also shown by the low neighborhood hit of 53.9%.

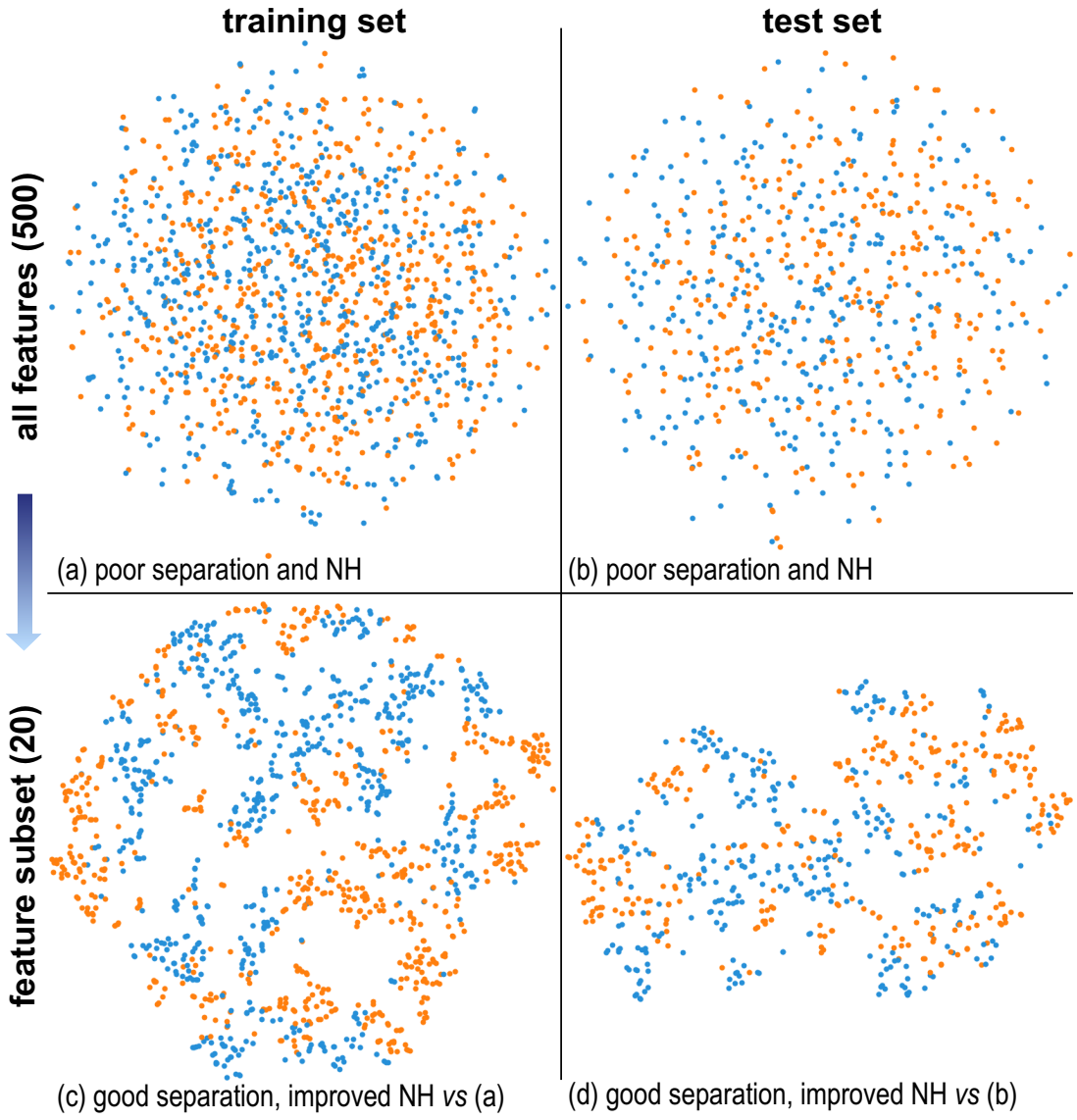


Figure 4.2: *Madelon* dataset. (a) Training set (NH: 53.9%). (b) Test set (NH: 50.97%). (c) Training set, feature subset (NH: 83.56%). (d) Test set, feature subset (NH: 74.15%).

If our projection is representative of the distances in the high-dimensional space, it is natural to interpret Fig. 4.2a as evidence that the classification problem is hard, at least if the learning algorithm being used is based on distances. We will show that, for this example, this observation holds even for learning algorithms that do not directly work with distances in the high-dimensional space. This characteristic is crucial if we want to

use projections as visual feedback about the quality of classification systems that use such algorithms.

Figure 4.2b shows the projection of the test data, which also has a low neighborhood hit (NH) and poor separation. Following the experimental protocol outlined in the previous section for hyperparameter search, consider the best (in terms of average cross-validation accuracy) classifier for each learning algorithm. If the hypothesis about the difficulty of this classification task is true, the expected result would be a low accuracy on the test data.

Figures 4.3a and 4.3b show the classification results for KNN (54.94% accuracy) and RFC (66.17%). The SVM classifier achieved 55.84% accuracy (not shown for brevity). Triangles in the scatterplots show misclassified observations, colored based on their misclassification. The accuracies on the test set are considerably low, and both KNN and SVM perform close to random guessing.

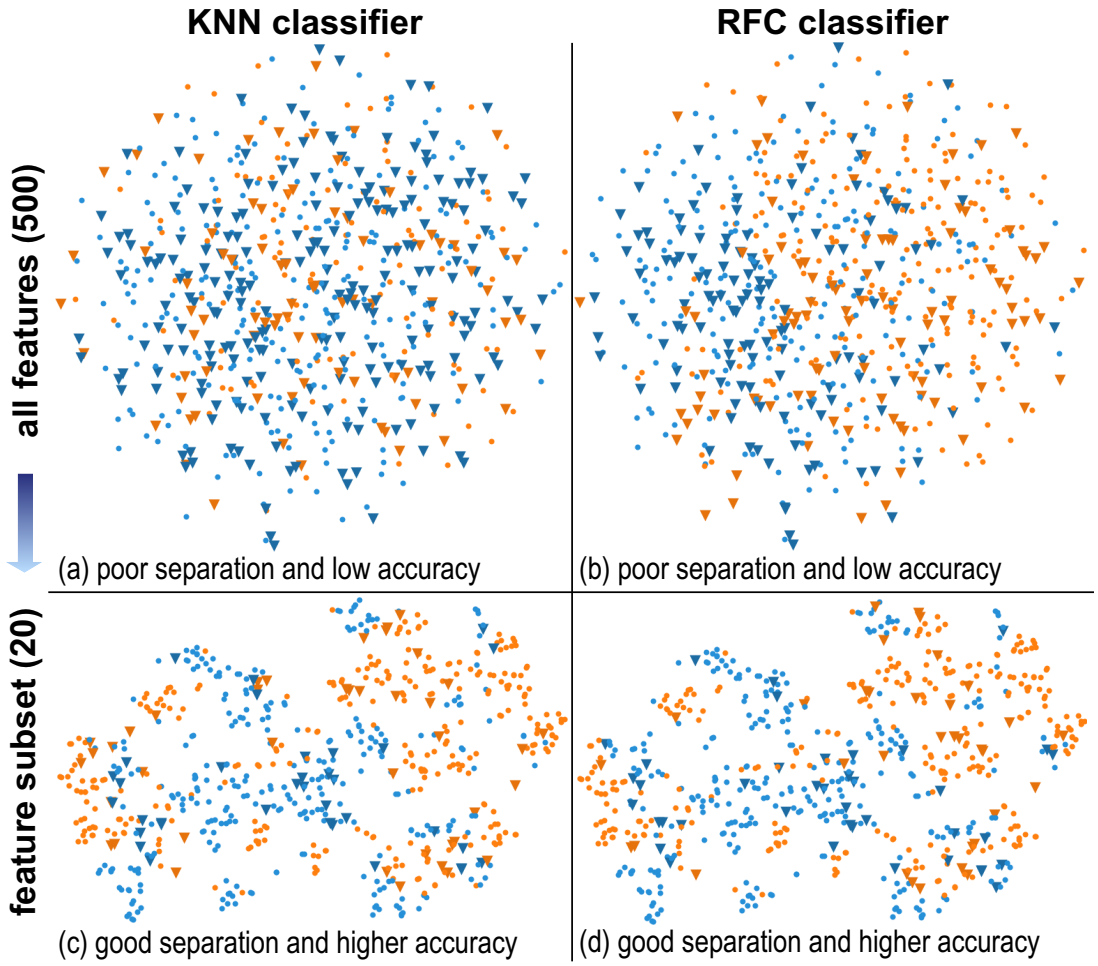


Figure 4.3: *Madelon* classification. (a) KNN (AC: 54.94%). (b) RFC (AC: 66.17%). (c) KNN, feature subset (AC: 88.62%). (d) RFC, feature subset (AC: 88.92%).

Goal 2: Although these results show that the poor visual separation is correlated to a low classification accuracy, nothing we have shown so far tells that good separation relates to high accuracy. Let us investigate this next, specifically showing how we can select an

appropriate subset of features to obtain good class separation.

Using extremely randomized trees as a feature scoring technique, consider a subset containing 20 of the original 500 features, chosen based on their discriminative power in the training set. In other words, we chose the best features $\mathcal{F}' \subseteq \mathcal{F}$ to separate the two classes in the high-dimensional space. Figure 4.2c shows the projection of the training set restricted to these features. Compared to the previous projection of the training set (Figure 4.2a), the NH has improved considerably, and the visual separation has also improved. This visual feedback gives evidence that the classification task may become easier using a feature subset.

Figure 4.2d shows that feature selection also enhances the visual separation of the test set. Therefore, the visual separation after feature selection *generalizes* well to the test data.

The final question is whether the good visual separation corresponds to higher accuracy in the test set. Figures 4.3c and 4.3d confirm this hypothesis. Notice that, after feature selection, both learning algorithms have greatly improved their results on the test set, with an increase of nearly 34% for KNN and 22% for RFC. In comparison, the neighborhood hit increased by almost 24% for the test set, and by almost 30% for the training set. A similar increase happens in the case of the SVM, which goes from 55.84% to 86.68% test accuracy after feature selection. In other words, as could be expected, removing irrelevant features considerably enhances the generalization capacity of the learned model.

Even more interestingly, after feature selection, we see that the misclassified observations in the test set are often surrounded by points belonging to a different class (see triangular glyphs in Figs. 4.3c and 4.3d). Thus, these observations could be interpreted as *outliers* according to the projection. Such feedback is *hard* to obtain from a traditional machine learning pipeline, and is valuable for understanding classification system malfunction. Manually inspecting misclassified observations and their neighbors without the help of visualization would be very time-consuming, and would not convey nearly as much insight about the *structure* of the data. Alternatives such as confusion matrices, for example, are difficult to interpret even for a modest number of classes (a confusion matrix for a 10-class problem has 45 independent values). The feedback presented by projections can, for example, prompt the users to consider special cases in their feature extraction pipeline.

Findings: In summary, the use case presented in this section shows how projections can predict classification system efficacy. In this use case, poor visual separation matches low classification accuracy, and good visual separation matches high classification accuracy. Furthermore, points that appear as outliers in a projection are often difficult to classify correctly.

4.4.3 Melanoma dataset

Data: The *melanoma* dataset contains 369 features extracted from 753 skin lesion images, which are part of the EDRA atlas of dermoscopy [5]. Class labels correspond to benign skin lesions (485 images) and malignant skin lesions (268 images). Note the considerable class unbalance in favor of the benign lesions.

The feature extraction process is described by Feringa [46], and involves interactive segmentation using the superpixel-based method introduced in Chapter 3. This dataset was developed partially as a highly challenging application for the approach proposed in this chapter.

Goals: The main goal of the experiments performed using this real-world dataset is to show the type of feedback that can be obtained through projections when the classification problem is difficult and the visual class separation is poor.

Figure 4.4a shows the projection of the training data. We see that the separation between classes is poor, which is confirmed by a low NH. Consider the set of 20 best features to discriminate between the two groups in the training set, according to extremely randomized trees. The corresponding projection of the training data restricted to these features is shown in Fig. 4.4c. Arguably, the separation is slightly improved, which is confirmed by a higher NH value.

Figure 4.4: Melanoma dataset. (a) Training set (NH: 64.87%). (b) Test set (NH: 62.35%). (c) Training set, feature subset (NH: 72.38%). (d) Test set, feature subset (NH: 62.55%)

Figures 4.4b and 4.4d show the projections of the test data before and after feature selection, respectively. The poor separation is confirmed in the test data. More importantly, the separation does not seem to be better in the test set after feature selection. In other words, feature selection does not appear to have generalized particularly well to

the unseen (test) data. From this evidence, we naturally suspect that classification accuracy is poor, and that feature selection will not enhance accuracy. Our next experiments confirm this suspicion.

Figure 4.5a displays the classification results on the test set obtained by the most effective learning algorithm (SVM, according to our protocol), using all the features. The class unbalance of the data places the expected accuracy of always guessing the most frequent class at 64%. Hence, an accuracy of 77.69% is not quite satisfactory. KNN also performs poorly, achieving only 73.71% accuracy (Fig. 4.5b). This is evidence that the classification task is hard.

Figure 4.5: Melanoma classification. (a) SVM (AC: 77.69%). (b) KNN (AC: 73.71%). (c) SVM, feature subset (AC: 74.9%). (d) KNN, feature subset (AC: 77.69%). The uniformity of blue classifications in the center of the projections shown in (c) and (d) confirms that distances in the projection are good indicators of classifier behavior.

Figures 4.5c and 4.5d show the classification results obtained after feature selection. As we see, feature selection improved the efficacy of the KNN classifier (from 73.71% to 77.69%) to the same level as an SVM using all features. On the other hand, the SVM results deteriorated after feature selection.

Furthermore, notice the uniformity of blue classifications in the center of the projections shown in Figs. 4.5c and 4.5d. This confirms that distances in the projection are

good indicators of classifier behavior in this case, even when the learning algorithm does not directly use distances in the original high-dimensional feature space (Fig. 4.5c).

As anticipated, feature selection did not improve generalization efficacy. Even so, reducing the number of features to approximately 5% of the original has benefits in computational efficiency and knowledge discovery. The reduced set of features contains valuable information to the system designer, and indicates characteristics of the problem where designers may decide to focus their efforts. In other words, the use of feature selection, while not directly improving classification system accuracy, added value by reducing costs through data reduction.

4.4.4 Corel dataset

Data: The *Corel* dataset contains 150 SIFT features extracted from 1000 images by Li et al. [92]. Class labels correspond to 10 image types: Africa, beach, buildings, buses, dinosaurs, elephants, flowers, horses, mountains, and food. The dataset is perfectly balanced between classes.

Goals: This experiment shows that projections can give insight into class-specific behavior, and also provides more evidence that projections can predict classification accuracy.

Figures 4.6a and 4.6b show projections of the training and test data, respectively. Except for a *confusion zone* between the classes marked as green, orange, yellow and brown, both projections show well-separated clusters. This separation is confirmed by a high NH value in both cases.

These projections can be interpreted as evidence that the classification task is easy. Confirming this hypothesis, Fig. 4.7a shows the classification results for the best classifier (RFC). As expected, the accuracy obtained is very high (91.81%), considering that this is a balanced 10-class problem. More interesting, however, is the fact that many classification errors occur in the confusion zone observed in the projection of the test set. Thus, conclusions drawn from the visual feedback about confusion zones in the training set do generalize to unseen (test) data. Notice that the concept of confusion zone is only possible because the data are *spatially* represented. It is, to our knowledge, not possible to depict a confusion zone otherwise. This is another valuable characteristic of our proposed projection-based representation.

We also use this dataset to consider an alternative scenario for predicting system efficacy. This scenario shows, again, that projections are reliable predictors of classification system behavior. Consider the best 10 features to discriminate class 4 (purple) from other classes, according to extremely randomized trees. The projection of the data restricted to this set of features is shown in Fig. 4.6c. As expected, note how class 4 is very well separated (center left), while observations in the other classes are poorly separated from each other. This is confirmed by low NH values (28.68%) and perfect *binary* NH values, when class 4 is considered against the rest. Figure 4.6d confirms that this characterization generalizes to the test data.

The poor separation between classes other than 4 leads us to expect poor accuracy results. Figure 4.7b shows the classification results using the features selected to separate class 4 from the rest, in the multi-class problem, which confirm this expectation. In

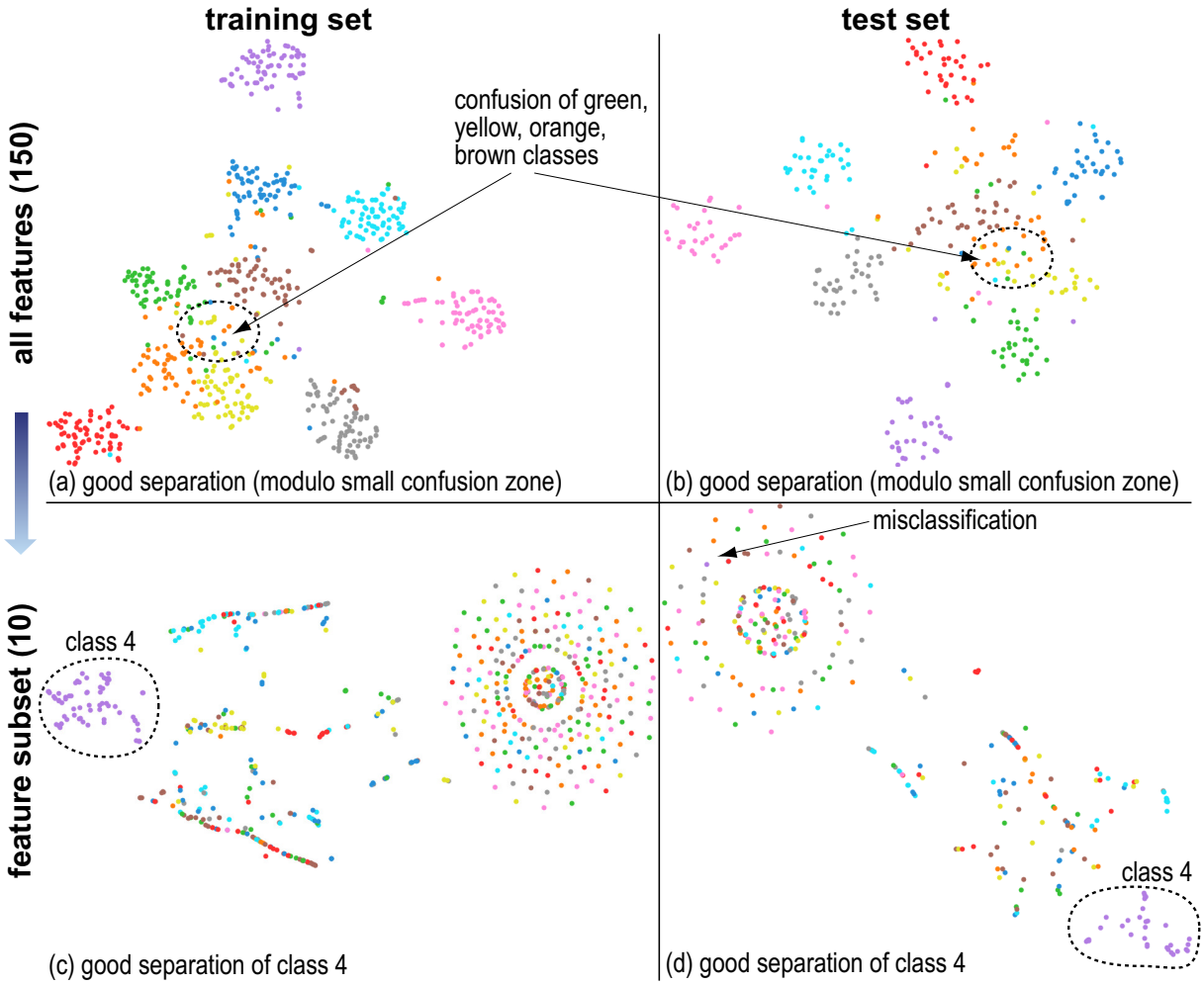


Figure 4.6: Corel dataset. (a) Training set (NH: 85.7%). (b) Test set (NH: 82.73%). (c) Training set, feature subset (NH: 28.68%, 4 vs rest NH: 100%). (d) Test set, feature subset (NH: 22.18%, 4 vs rest NH: 99.34%).

contrast, the binary classification accuracy is *almost perfect* (99.7%, image omitted for brevity). There is a single mistake in the binary classification, which is placed in the top left corner of the projection (top left of Fig. 4.6d). The projection was also able to predict the existence of this outlier.

4.4.5 Parasites dataset

Data: The *parasites* dataset contains 9568 observations and 260 classical image features extracted from (pre-segmented) objects in microscopy images of fecal samples [138]. We restricted ourselves to a subset of the original data that contains only the protozoan parasites and impurities (objects that should be ignored during analysis). Almost sixty percent of the observations correspond to impurities, which gives a significant class unbalance.
Goal: We present here one last example of the predictive power of projections, using a medium-sized realistic dataset. In this case, the projection reveals the presence of a large number of confounding observations that, when removed, increase classification accuracy.

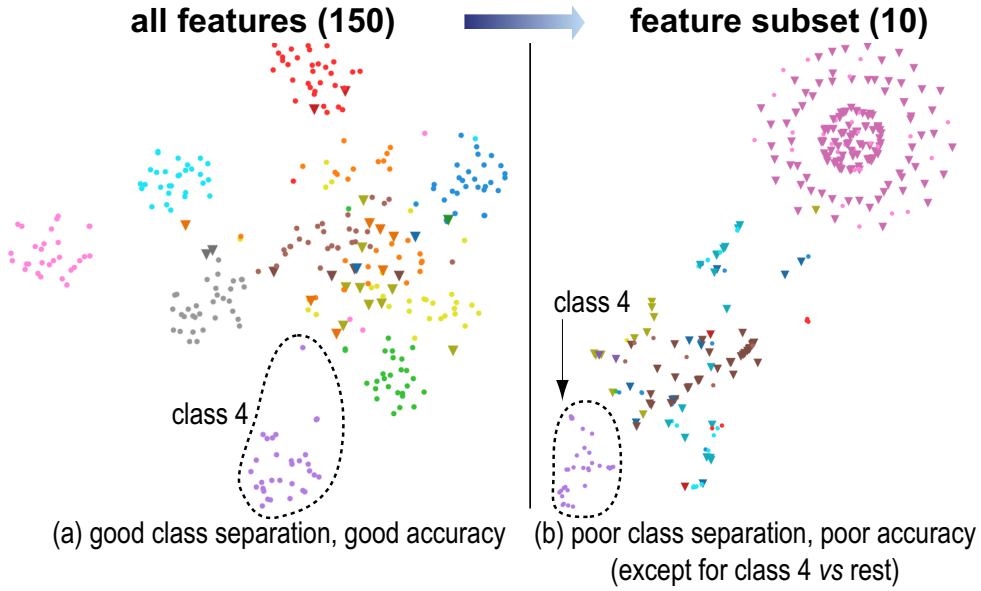


Figure 4.7: Corel classification. (a) RFC (AC: 91.81%). (b) RFC, feature subset (AC: 34.55%, 4 vs rest AC: 99.7%).

Figure 4.8a displays the projection of the training set. We immediately see that impurities (marked pink) spread over almost the entire projection space. This is also seen in the projection of the test set (Fig. 4.8b). In other words, we have weak evidence that the impurities may be confounded with almost every other class.

Figures 4.8c and 4.8d show the projections of the training and test data, respectively, when the impurities are removed from the data. Therefore, the other classes be reasonably well separated from each other when impurities are ignored.

Considering again all observations, Figure 4.9a shows classification results for the best classifier (SVM, according to our protocol). Given the perceived poor visual separation, this result may be considered surprisingly good, which shows that perceived confusion is not definitive evidence. In Section 4.6, we will show an extreme example of this behavior. In a number of cases, however, we have seen that the evidence is much stronger in the other direction: when the perceived visual separation between classes in a projection is good, the classification results are also good.

Consider next our dataset restricted to all the classes except impurities. Figure 4.9d shows KNN classification results, which are improved from 82.29% to 89.49% accuracy. However, SVM results are not significantly improved in this restricted task (approximately 2% accuracy increase). Once again, note how the confusion zones contain the majority of misclassifications. Apparently, the SVM learning algorithm is able to deal better with the confusion between impurities and parasites. In this case, the projection was better to anticipate the behavior of the distance-based learning algorithm.

This is the largest dataset considered in our experiments. Note that the projections of the training and test sets are somewhat similar (*e.g.*, Figs. 4.8c and 4.8d). This highlights the importance of using representative datasets to study a problem using projections.

The difficulty of separating impurities from other classes could also be diagnosed from

a confusion matrix. In practice, this insight could be used by the designer to study the classification of impurities as a separate problem. However, projections provide a more compelling visual representation of the same phenomenon, allowing the designer to inspect the observations in confusion zones. Such spatial information about relationships is lost in a confusion matrix.

4.4.6 Conclusion

The experiments performed for the four datasets in this section support our claim that projections provide useful visual feedback about the ease of designing a good classification system. This visual feedback helps finding outliers, overall separation between observations in distinct classes, distribution of observations of a given class in the feature space, and presence of neighborhoods with mixed class labels. Arguably, the first two tasks have the most well-developed traditional feedback mechanisms: outlier detection, manual misclassification inspection, efficacy measures, and confusion matrices. The qualitative nature of the last two tasks makes them more difficult. This makes a strong case for the use of projections, even if there is no hard guarantee that the visual feedback offered by projections is definitely helpful for a given dataset.

4.5 T2: Improving system efficacy

The previous section showed how projections can be useful for predicting classification system behavior. If a particular system performs well, there is no further effort required from the system designer. Instead, consider a classification system that generalizes poorly to unseen data. Because the design space (feature descriptors, learning algorithms and hyperparameters) is immense, designers can benefit from insightful feedback about their choices. In that case, we have also shown that qualitative feedback from projections can be highly valuable.

Building on the use of projections for the first task (**T1**), this section focuses on the use of projections for the task of improving classification system efficacy (**T2**). In section 4.5.1, we present a visual feedback methodology that enables **T2**. In Sections 4.5.2 - 4.5.5, we describe use cases that employ this methodology. Finally, Section 4.5.6 summarizes these examples by presenting a workflow for our proposed classification system improvement process.

4.5.1 Proposed methodology and tooling

Our methodology for classification system improvement through interactive projections is implemented into a tool composed of six linked views (Fig. 4.10), as follows.

The **observation view** shows the image associated to each observation $\mathbf{x}$ in the dataset $\mathcal{D}$, if any, which are optionally sorted by a feature of choice. This provides an easy way to verify if a feature corresponds to user expectations.

Available in `http://www.cs.rug.nl/svcg/People/PauloEduardoRauber-featured`.

The **feature view** shows all features $\mathcal{F}$, optionally organized as a hierarchy based on semantic relationships. Within this view, users can select a feature-subset $\mathcal{F}' \subseteq \mathcal{F}$ to further explore.

The **group view** allows the creation and management of observation groups by direct selection in the observation view or in the observation projection view (discussed next).

The **observation projection view** shows a scatterplot of the projection of $\mathcal{D}_{\mathcal{F}'}$, the dataset composed of all observations restricted to the currently selected feature subset $\mathcal{F}'$. Points can be colored by a user-selected characteristic (such as class label or feature value), and are highlighted to show the selected set of observations.

Figure 4.10 also illustrates lensing, which optionally displays secondary characteristics on a neighborhood. In this particular case, the secondary characteristic is classification outcome (correct classifications in blue, incorrect in red).

The **feature scoring chart** ranks the features in $\mathcal{F}'$ by a relevance metric chosen by the user. We provide a variety of feature scoring techniques, including extremely randomized trees (Sec. 2.4.4, which we also employed in Section 4.4) [51], randomized logistic regression (Sec. 2.4.2) [103], SVM recursive feature elimination (Sec. 2.4.3) [62], and others. The feature scoring view also allows the user to select a subset of $\mathcal{F}'$ through interactive rubber-banding.

The **feature projection view** presents one point for each feature in $\mathcal{F}$. Features are placed in 2D by a technique that tries to preserve the structural *similarity* between features. For our purposes, we define the dissimilarity $d_{i,j}$ between features i and j as $d_{i,j} = 1 - |r_{i,j}|$, where $r_{i,j}$ is the (empirical) Pearson correlation coefficient between features i and j. This dissimilarity metric captures both positive and negative linear correlations between pairs of features, although it has shortcomings that we already discussed in Sec. 2.1. The dissimilarity matrix, which contains the dissimilarity between all pairs of features, can be represented in two dimensions by a *projection*, which is analogous to the projection of observations. As already mentioned in Sec. 4.2, similar visualizations already exist in the literature [158, 144]. However, we combine the feature projection view with task-specific information in a novel manner, as shown in the next sections.

We chose (absolute metric) multidimensional scaling [17] to compute feature projections (Sec. 2.6.3). According to preliminary experiments, MDS presents more coherent relationships between features and classes than t-SNE, which is important in the next sections. This is probably due to the difference in goals between the two techniques: absolute metric MDS attempts to preserve (global) pairwise dissimilarities [17], while t-SNE is particularly concerned with preserving (local) neighborhoods [148]. Alternative (dis)similarity metrics between features are also available in the tool, including mutual information (Sec. 2.4.1), distance correlation [139], and Spearman's correlation coefficient [29]. The feature projection view provides a counterpart to the observation projection view, and enables several interactions that will be detailed in the next sections.

Our visual analysis tool is implemented in Python, and uses Numpy [149], Scipy [76], pyqt, matplotlib [70], scikit-image [150], scikit-learn [118], pyqtgraph, and mlpy [2].

The next sections describe how our tool can be used to support the task of classification system improvement based on visual feedback obtained from both observation and feature projections. For an overview of tool usage, see Section 4.5.6.

4.5.2 Madelon: relationship between relevant features

Goal: In this section, we illustrate how the feature projection view can be used to select features by considering relationships between relevant features. As already mentioned, feature selection is a major challenge in classification system design. In particular, insight into the feature space can be very valuable when hand-engineered (off-the-shelf) features are used.

Consider a selection of the 20 best features to discriminate between the two classes of the Madelon dataset (Sec. 4.4.2) , performed using the feature scoring chart based on extremely randomized trees. The corresponding projections of observations and features are shown, respectively, in Figs. 4.11a and 4.11b. Each feature in the feature projection view is colored according to its relevance score (darker colors represent higher relevance according to extremely randomized trees). The 20 selected features are outlined in black. Note that the most relevant selected features (darker colors) are placed near the center of the feature projection, except for the least relevant one. This finding is notable, since the feature projection is created without any information about feature scores. This shows that relevant features are related (according to the feature dissimilary and relevance scoring metrics) in this dataset. Note that, in general, relevant features are not necessarily related. For instance, a feature can simply *complement* the discriminative role of other features.

Showing the relationships between feature scoring and feature similarity is a main asset of the feature projection view. Figures 4.11c and 4.11d show how such insight can be used: by removing the *outlier* feature (*i.e.*, the feature that is apparently unrelated to the rest of the selection), visual separation is preserved. In other words, the feature projection view let us *prune* the feature space while maintaining the desired visual separation (and NH), thereby reducing the size of the data that needs to be considered next.

Improvement: Table 4.1 presents the results of each learning algorithm on the Madelon test set, following the protocol described in Sec. 4.4.1, before and after removing the outlier feature mentioned above. As conveniently anticipated by the observation projection of the training set (Fig. 4.11c), the classification efficacy is maintained (and perhaps slightly improved). In summary, the feature removal suggested by the feature projection view has reduced the data size, but maintained classification accuracy.

Table 4.1: Madelon test set accuracies, feature selection according to Fig 4.11.

Features/Algorithm	KNN	RFC	SVM
20 features	88.62%	88.92%	86.68%
19 features	88.92%	88.92%	89.22%

4.5.3 Corel: class-specific relevant features

Goal: This section shows how the feature projection view can be used together with the observation projection view to find class-specific relevant features, using the Corel dataset (Sec. 4.4.4) as an example. When improving system efficacy, such information is useful both for feature selection and for understanding classification system behavior.

We already showed (Sec. 4.4.4) that we can choose features that are good to discriminate one of the classes in the Corel dataset (class 4, which corresponds to dinosaur drawings), while making discrimination between the other classes very difficult. Figures 4.12a and 4.12b show the corresponding observation and feature projections. Once again, we see that the discriminative features are highly related.

Consider an analogous feature selection aimed to discriminate class 3 (bus pictures) from the other classes. Figures 4.12c and 4.12d show the corresponding projections. Comparing the feature views (Figs. 4.12b and 4.12d), we easily see that the sets of powerful discriminative features for the two classes are disjoint. This information could not be easily obtained from the feature scoring bar chart mentioned in Section 4.5.1, since features are generally difficult to locate in that visualization. As inspecting the precise ranking of each feature is easier in the bar chart, the two views are complementary. These interactions require very little effort from the user, who can inspect several feature combinations in a few minutes.

If the user is interested in a rough estimate of classification efficacy, our tool can also compute and display classification results (for a chosen learning algorithm) based on k-fold cross-validation. This process partitions the current data into k disjoint validation sets, and a classifier trained on the rest of the data is used to classify each validation-set. Classification results for the distinct validation-sets are aggregated and displayed, leading to images similar to Fig. 4.7. These representations do not replace proper evaluation in a held-out test set (as in Sec. 4.4), but are useful feedback sources during the interactive feature analysis process.

Improvement: Table 4.2 presents the result of each learning algorithm on the Corel test set, following the protocol in Sec. 4.4.1, for the task of discriminating classes 3 and 4 from the rest (*i.e.,* classes 3 and 4 are treated as a single class in a binary classification task), for all features and the subset of 26 features that were considered (separately) relevant for classes 3 and 4. As predicted by the observation projections of the training set shown in Figs. 4.12a and 4.12c, the classification efficacy is preserved. In summary, our visual analysis methodology allowed us to prune the feature space from 150 to only 26 features, and construct a binary classifier for classes 3 and 4 *vs* rest that has the same quality as a classifier that uses all features.

Table 4.2: Corel test set accuracies, classes 3 and 4 vs rest, relevant features according to Fig. 4.12.

Features/Algorithm	KNN	RFC	SVM
All (150) features	98.18%	98.79%	98.48%
26 features	98.48%	98.79%	98.79%

4.5.4 Melanoma: alternative feature scores

Goal: The joint display of feature similarity and relevance is useful in other ways, as shown next. Here, our representation enables comparing the results of different feature scoring techniques. Since the techniques are based on distinct principles, comparing their

results to find features that are consistently considered effective is a valuable task for improving system efficacy.

Consider the feature projection view of the melanoma training set (Sec. 4.4.3) shown in Fig. 4.13a. As usual, colors represent the relevance of each feature to discriminate between the two classes present in the dataset (according to extremely randomized trees). We see a concentration of relevant features between the center and the bottom right. Again, the feature placement reinforces the feature scoring information. The presence of *zones* of highly relevant features is highly suggestive for the exploration of the feature space, as shown in Sec. 4.5.2.

Consider an alternative feature (relevance) scoring obtained by another technique – in this case, randomized logistic regression [103] (Sec. 2.4.2) – shown in Fig. 4.13b. We see that the distribution of relevancies is very different according to the second technique, which places higher cumulative relevance into fewer features. However, note that the two techniques agree on the irrelevance of the features in the bottom right and top left. This visual metaphor, where similar features are placed near each other, is a natural way to display such information.

The image features in this dataset have meaningful names, which can be inspected by hovering over the points. Using this mechanism, we find that the irrelevant peripheral points correspond mostly to histogram bins that have little (or even zero) variance across all images in our dataset. As expected, these features have almost no predictive power.
Improvement: Table 4.3 presents the result of each learning algorithm on the Melanoma test set, following the protocol in Sec. 4.4.1, for all 369 features and the 58 (mostly) relevant features shown in Fig. 4.13a and 4.13b. Although the KNN and SVM results deteriorated slightly, the RFC result improved. Also, our method allowed us to discard a significant number of hand-engineered features. Besides saving significant time in feature extraction, the insight provided by our visual analysis of the feature space helps in deciding which types of features are most relevant for classification.

Table 4.3: Melanoma test set accuracies, relevant features according to Fig. 4.13

Features/Algorithm	KNN	RFC	SVM
All (369) features	73.71%	76.49%	77.69%
58 features	73.31%	77.29%	76.10%

4.5.5 Parasites: importance of projection error measures

Goal: In this section, we use the observation projection view to focus on a different kind of visual feedback. Our tool also presents the *aggregate projection error*, a per-point metric of distance preservation after DR [101]. Intuitively, a point has a high aggregate error when its corresponding high-dimensional distances to the other observations are poorly represented by the low-dimensional distances in the projection. This feedback about the quality of a given projection is key to our methodology. We illustrate this by a simple use case, where an *interesting* observation is highlighted by its projection error.

Consider once again the parasites dataset restricted to non-impurities (Sec. 4.4.5). Figure 4.14a shows the aggregate error for the test set (higher errors in darker colors).

We see a point near the center of the projection with a relatively high aggregate error (square in Fig. 4.14a). As colors map relative errors, this does not necessarily mean that the absolute aggregate error is high. Yet, this point is clearly an *outlier* in aggregate error when compared to its low-dimensional neighbors. In Fig. 4.14b, we see that the point is surrounded by points belonging to other classes. By our definition, this point is an outlier with respect to its positioning given its class label. Note that the aggregate error is computed without any information about class labels, and also draws attention to this particular observation.

One possible explanation for a high aggregate error is that the projection placed a point in a poor manner. The corresponding observation might not even be an outlier. In fact, the point is correctly classified by RFC and SVM, which weakly supports this hypothesis. However, KNN classified the point incorrectly (see inset in Fig. 4.14b). Therefore, it is still unclear whether this point is a true outlier. The aggregate error view was successful in focusing attention into an *interesting* observation, which warrants further inspection of its characteristics and features.

To enable similar feedback, our tool could potentially use several other error metrics and visual depictions of projection quality (*e.g.*, [101, 7]). As already mentioned in Sec. 4.4.1, such depictions are highly important for assessing how to interpret the visual feedback.

4.5.6 Proposed workflow

We now summarize the value added by the insights described in Sections 4.4 and 4.5 by revisiting the high-level workflow outlined in Fig. 4.1.

Our workflow begins when the user loads the data into our analysis tool and considers the observation projection. If the perceived class separation in this projection is good, the classification task is likely quite simple (as discussed in Sec. 4.4). As an extreme example, consider the projection of the Corel dataset, where even a 1-nearest neighbor algorithm in the 2D projection space would achieve good results. In such cases, the user can follow the traditional machine learning pipeline, with a high expectation that the system will perform well.

A more interesting scenario occurs when the perceived class separation in the projection is poor. In this case, the next step is to use the mechanisms provided by our tool to find a feature subset that brings separation. This may require several iterations of feature scoring, analysis, and backtracking. If no separation improvement can be found, there are two possible scenarios: classification efficacy is satisfactory (the projection is misleading with respect to classifier behavior) or unsatisfactory. The first case is easy to diagnose, and consists on conducting experiments following the traditional machine learning pipeline. The second case is the most complicated. In this case, we have shown that the qualitative aspects of our proposed visualizations are crucial in enabling the designer to diagnose the system. For this purpose, our tool provides mechanisms to detect the presence of outliers and confusion zones, and also to inspect classification results based on a visual metaphor that represents observations in a consistent way. By inspecting the observation projection, the designer receives visual feedback about which features are im-

portant to eliminate confusion between classes. Furthermore, using the feature projection view and feature scoring methods, the designer can reason about the discriminative power of features, and focus effort on related (or complementary) feature descriptors. The new alternatives devised during this analytic process can be fed back into the tool, closing the cycle.

4.6 Discussion

This section discusses several important aspects of our proposed methodology and experiments.

Coverage: As any experimental study, many conclusions are limited to the datasets that we presented. The particular random choice of training and test data also affects the results, although the amount of data we considered diminishes this concern. Importantly, the extent of our validation (*i.e.*, experimental protocol, number of datasets and learning algorithms) is in line with comparable works in visual analytics and machine learning.

While we have conducted less organized experiments in additional datasets, the four datasets discussed in this chapter illustrate well all types of feedback that can be obtained from projections. We also experimented with other dimensionality reduction techniques (namely, LSP [116] and LAMP [75]), but obtained the best predictive feedback from t-SNE [146]. Finally, our choice of learning algorithms for validation (KNN, RFC, SVM) considers their widespread popularity, and aims to make our approach appealing to a large number of practitioners. The positive results obtained with these highly distinct algorithms suggest that our approach is valuable for other learning algorithms.

Limitations: It is easy and instructive to construct a synthetic example where projections do not provide valuable visual feedback for classification system design, which we describe next.

Consider the task of classifying observations sampled from two 10-dimensional parallel (affine) hyperplanes that correspond to distinct classes. Consider also that the distance between these hyperplanes is small when compared to the expected distance between any pair of neighboring observations from the same hyperplane. By construction, this classification task is very easy for a linear SVM, which consistently obtains 100% accuracy following the experimental protocol detailed in Sec. 4.4.1. At the same time, a DR technique that tries to preserve the original distances in the high-dimensional space will not show a clear separation between the two classes, as shown in Fig. 4.15. In simple terms, the visual feedback is misleading, because the classification task is easy, but there is no apparent visual separation between classes. It is important to note that other learning algorithms did not perform well on this test set (KNN: 51.20%, RFC: 54.94%). However, we believe it is also possible to construct examples where the visual feedback is unhelpful for those algorithms.

Despite this worst-case behavior, we argue that the results presented in Secs. 4.4 and 4.5 support our claims that our proposed approach is highly valuable, particularly considering the very low investment necessary to explore data by our proposed methodology and tooling.

Scalability: Our feature space exploration approach benefits from the visual scalability of projections to thousands of high-dimensional observations and features, although visual clutter eventually becomes an issue for the quality of the visual feedback. The computational scalability limits are imposed by the requirement of interactive response times. The scenarios presented here can be explored in interactive time using a typical desktop computer. The main bottleneck consists on recomputing projections for different subsets of features. For some dimensionality reduction techniques [148], this issue becomes significant in datasets containing more than a few thousands of observations, while others are able to deal with hundreds of thousands of observations at interactive paces [116, 75].

4.7 Conclusion

In this chapter, we have shown that projections are useful tools for predicting classification system efficacy in several real and synthetic datasets. The visual feedback given by projections is especially helpful in qualitative tasks. These tasks include inspecting the presence of outliers, overall separation between observations in distinct classes, distribution of observations of a given class in the feature space, and presence of neighborhoods with mixed class labels.

We also introduced a methodology that uses projections as a basis for an interactive system designed to give insight into the feature space. This methodology, and associated tooling, can aid a classification system designer in improving classification efficacy. In particular, we showed how a projection representing observations can be integrated with an interactive representation of feature similarity to aid in this task.

Future works may integrate specific capabilities of some dimensionality reduction techniques into our methodology, such as control point positioning. Another worthwhile goal is providing visual support to semi-supervised learning tasks, such as active learning.

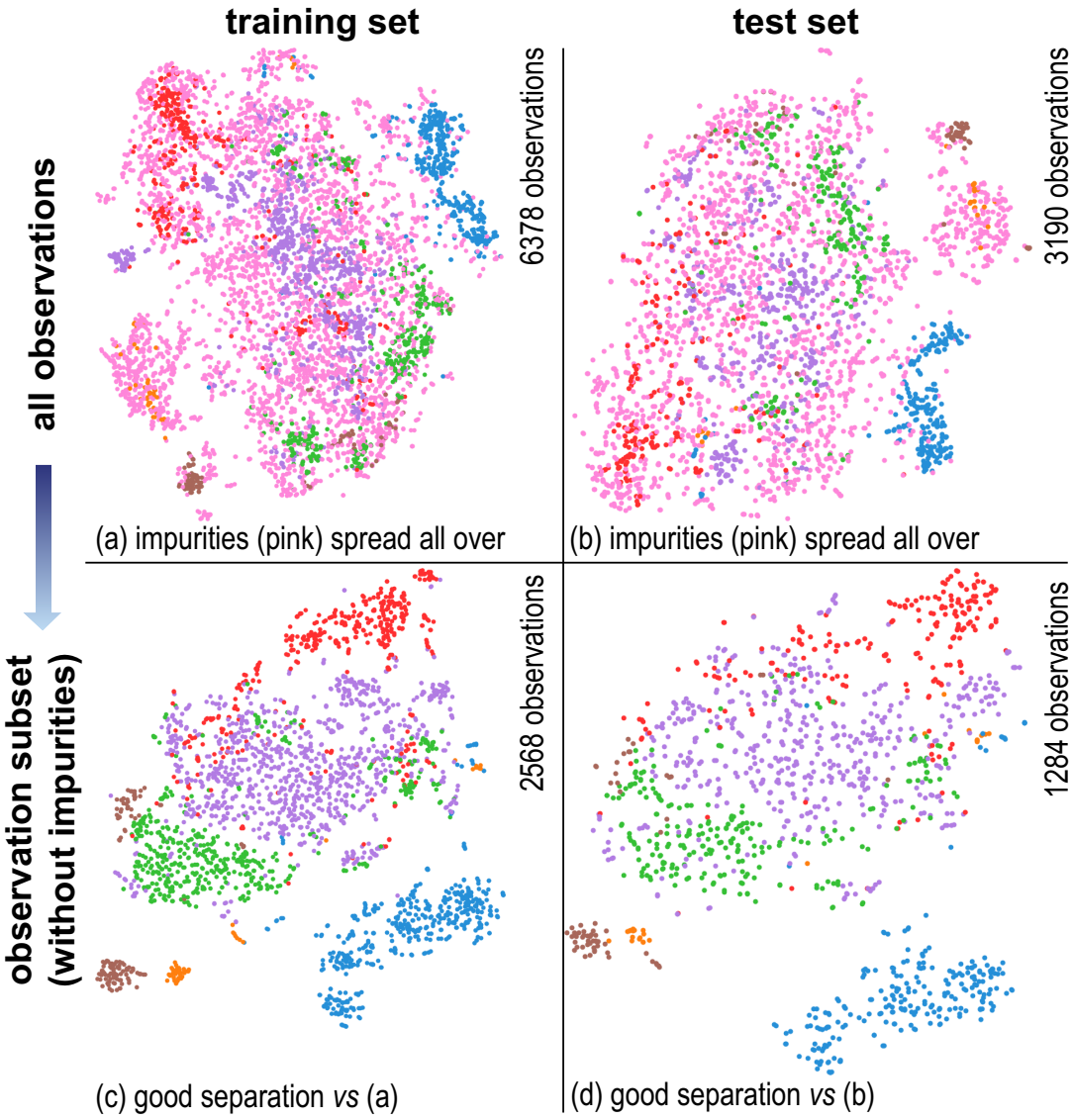


Figure 4.8: Parasites dataset. (a) Training set (NH: 74.35%). (b) Test set (NH: 68.49%). (c) Training set, observation subset (NH: 87.22%). (d) Test set, observation subset (NH: 82.31%).

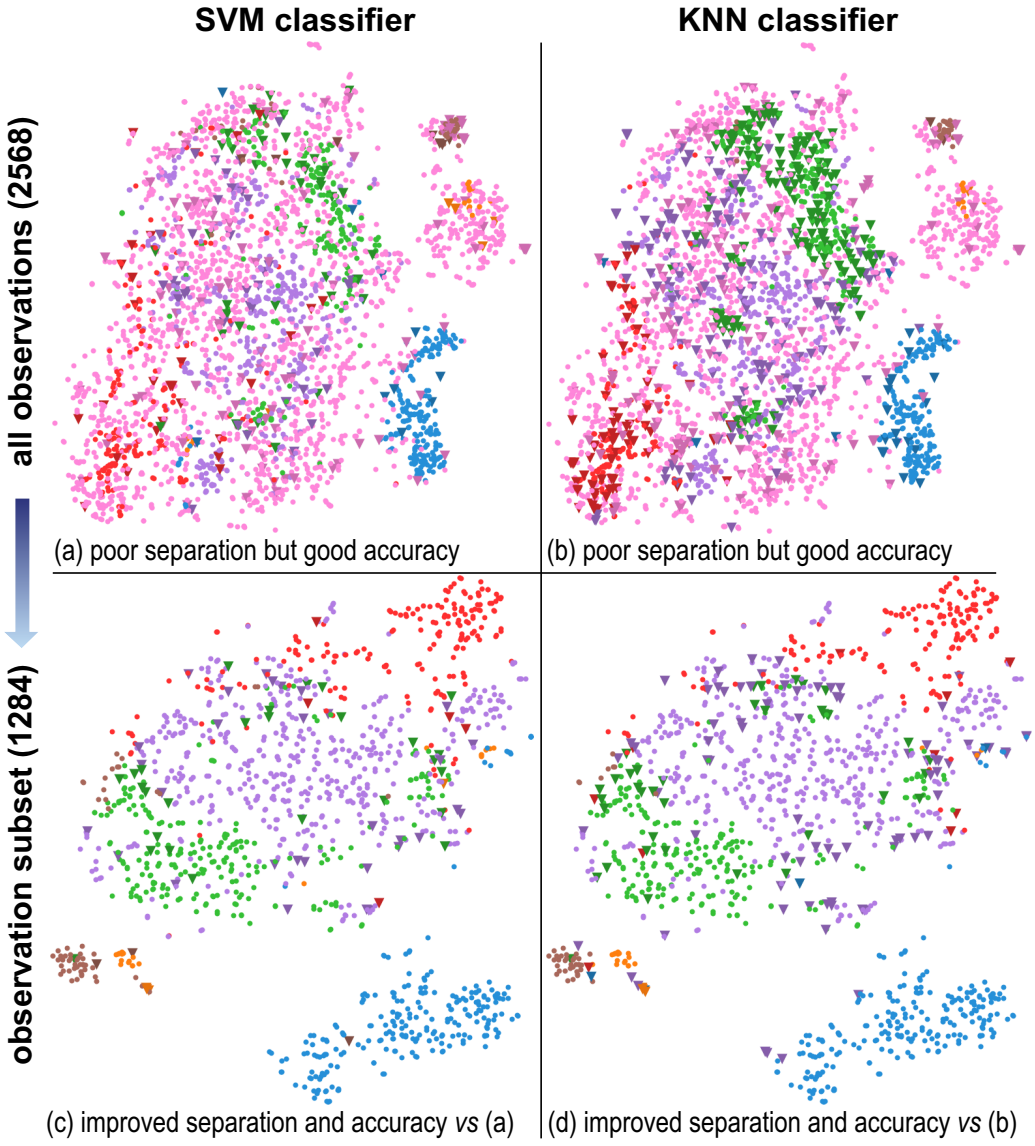


Figure 4.9: Parasites classification. (a) SVM (AC: 92.7%). (b) KNN (AC: 82.29%). (c) SVM, observation subset (AC: 94.55%). (d) KNN, observation subset (AC: 89.49%).

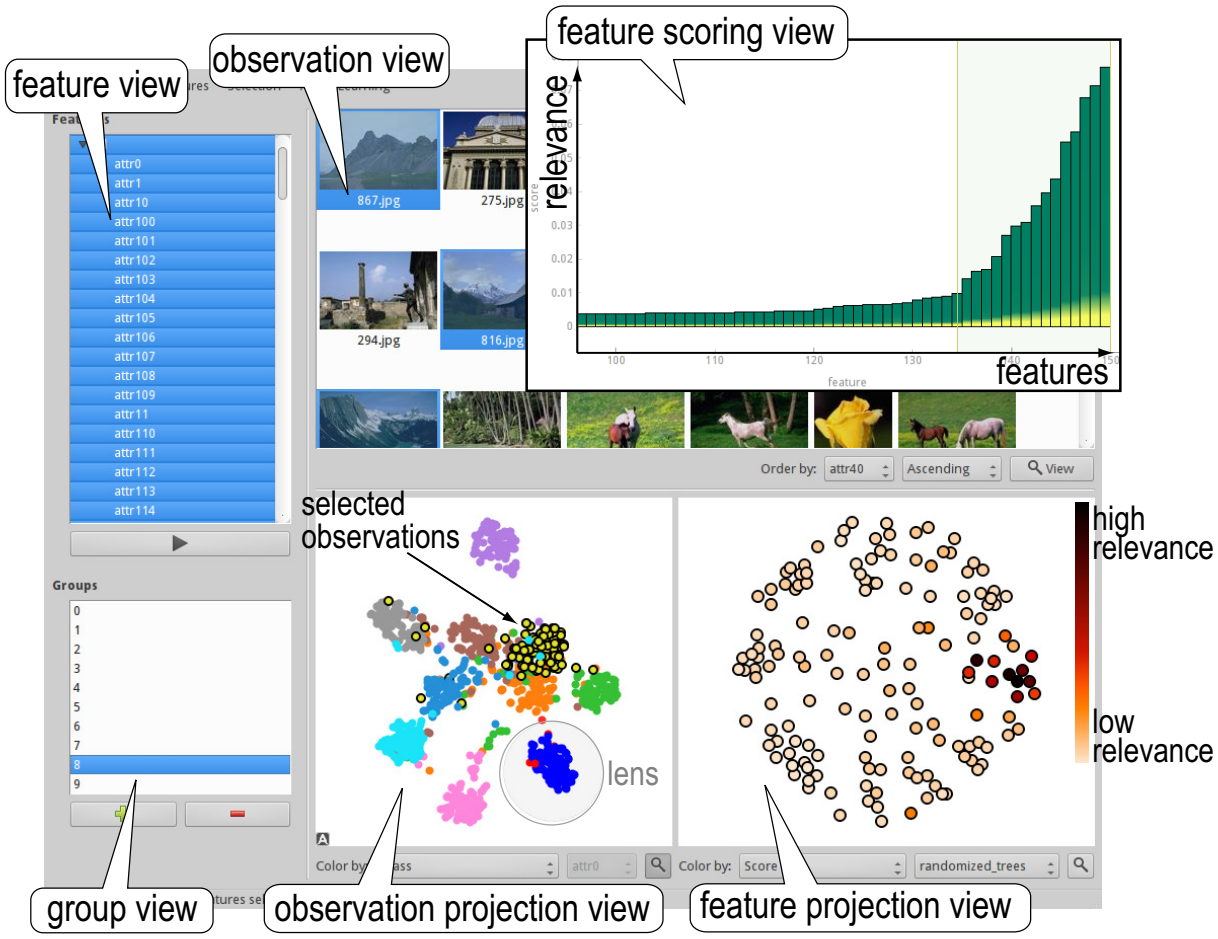


Figure 4.10: Feature exploration tool, showing the Corel dataset. Shows the observation view, feature view, group view, observation projection view (lensing observations, colored by classification; yellow observations are selected), feature scoring chart (showing best features to discriminate yellow class *vs* rest), and feature projection view (showing best features to discriminate yellow class *vs* rest, using a heat colormap).

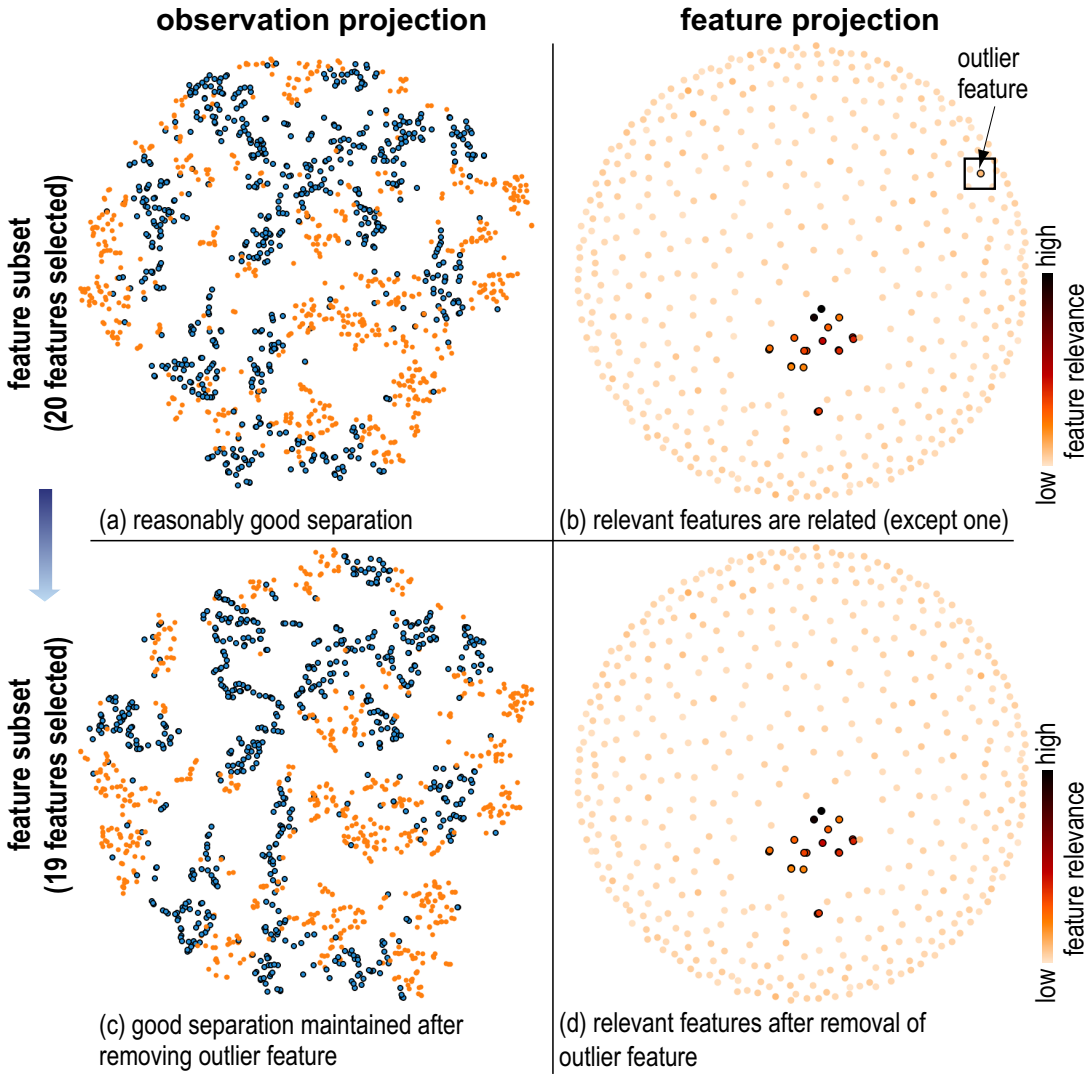


Figure 4.11: Madelon training set. (a,b) Observation and feature projections, 20 features selected (NH: 83.56%). (c,d) Observation and features projections, one less feature (NH: 84.55%).

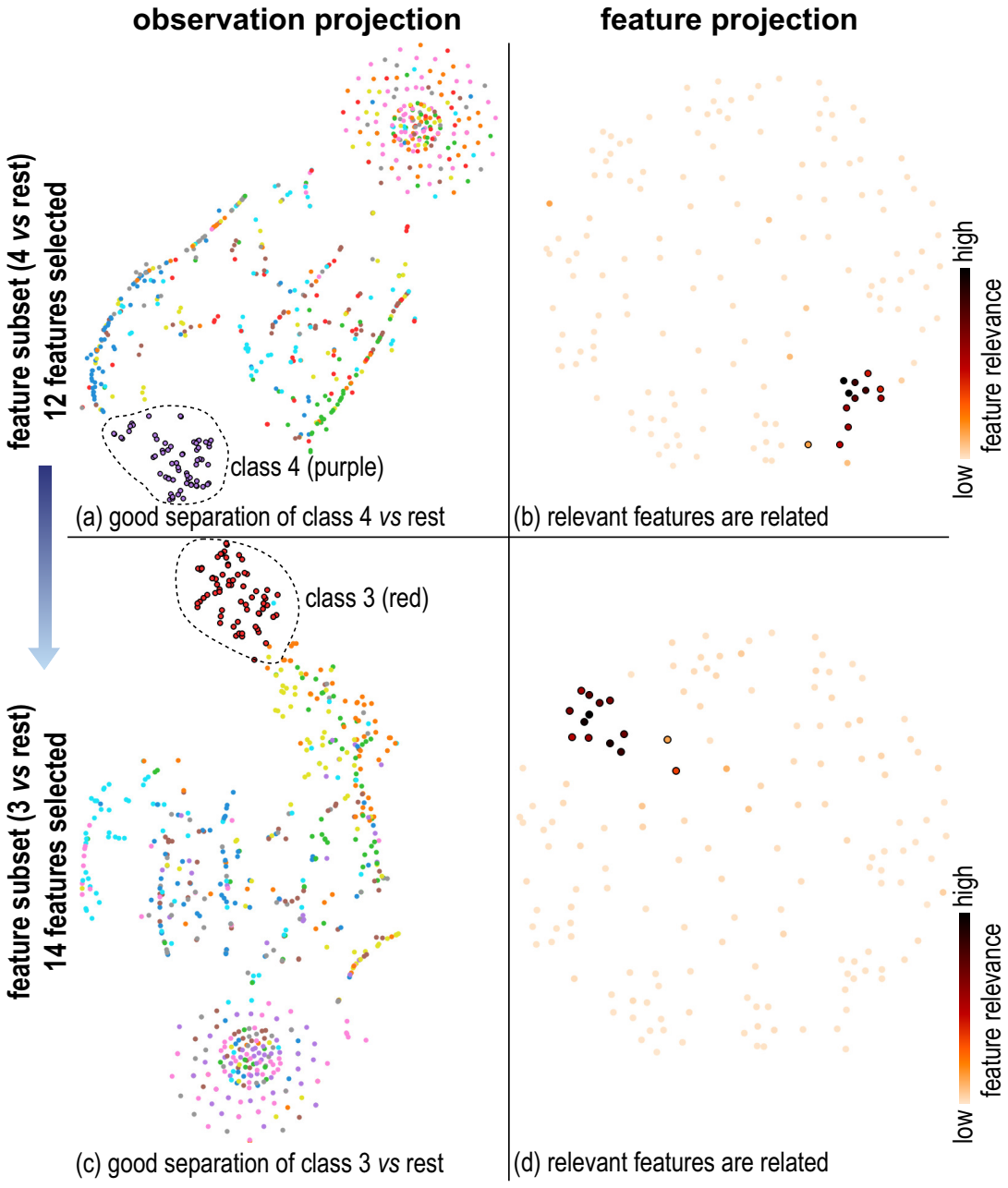


Figure 4.12: Corel training set. (a,b) Observation and feature projections, feature subset (4 vs rest, Binary NH: 99.73%). (c,d) Observation and feature projections, feature subset (3 vs rest, Binary NH: 99.25%).

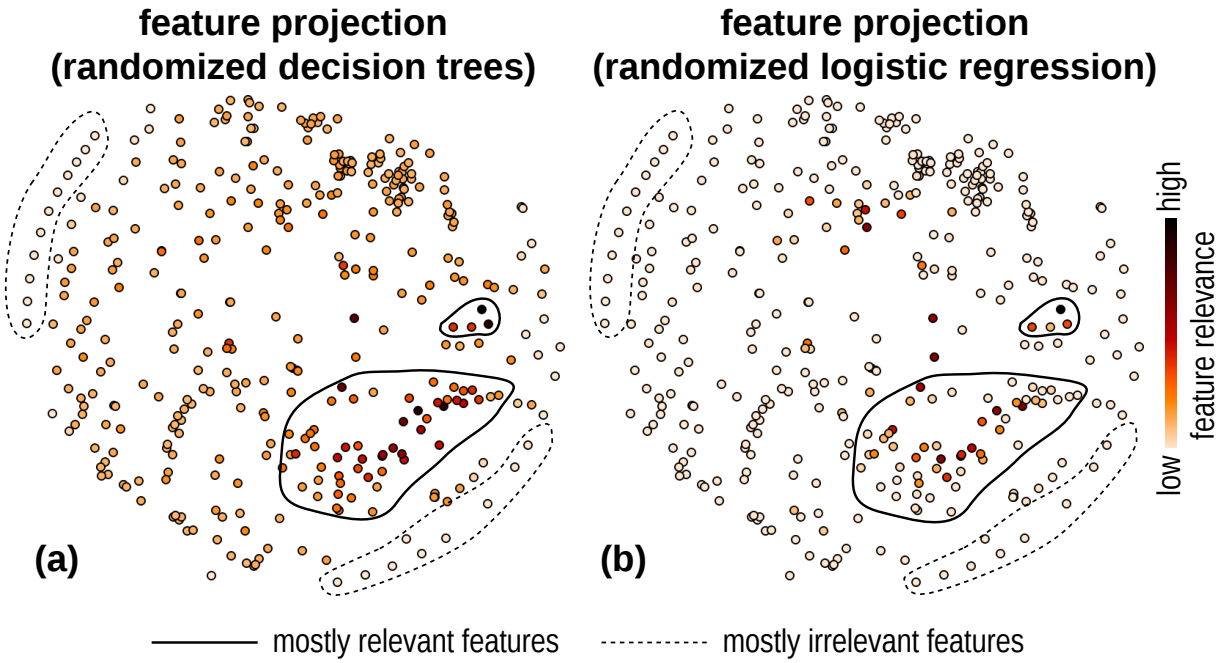


Figure 4.13: Feature projection for melanoma training set. (a) Feature scoring by randomized decision trees. (b) Feature scoring by randomized logistic regression

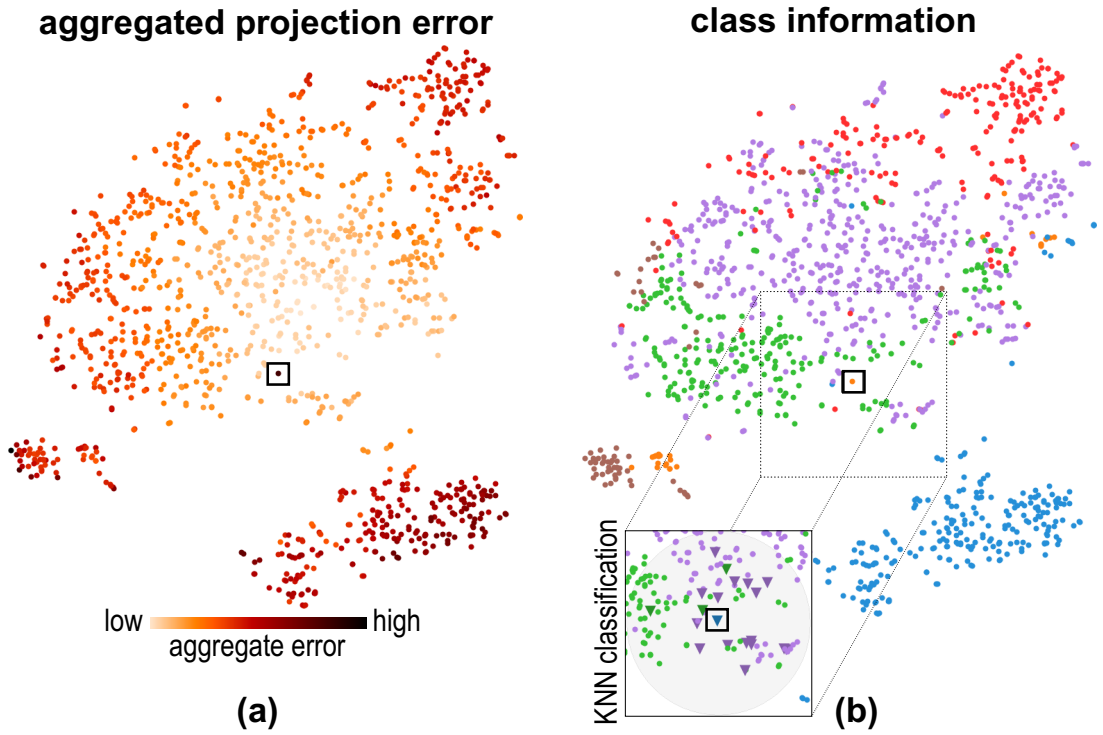


Figure 4.14: Parasites test set, observation subset. (a) Aggregate error. (b) Original classes, inset showing KNN classification.

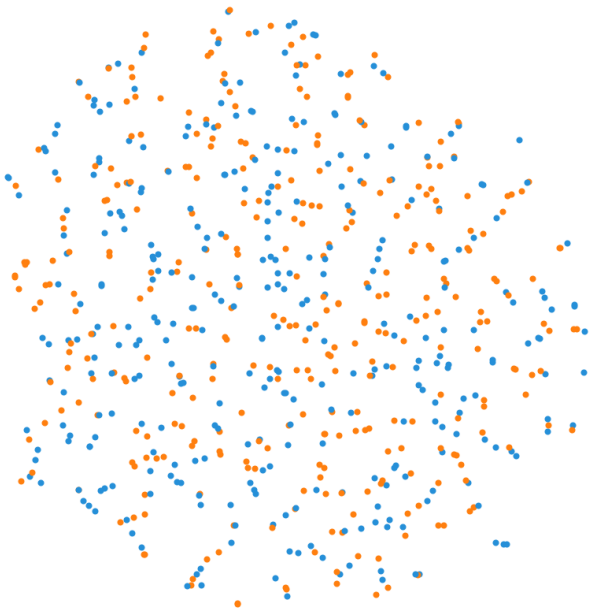


Figure 4.15: Planes classification, Linear SVM (AC: 100%).

Chapter 5

Visualizing artificial neural networks using projections

In Section 2.3.5, we mentioned that advances in computational power and techniques for building and training artificial neural networks (ANNs) have allowed these models to achieve state-of-the-art results in many applications related to pattern recognition [130]. However, successfully training ANNs is generally time-consuming, and requires significant expertise [12].

In this chapter, we demonstrate the potential of dimensionality reduction techniques to provide insightful visual feedback about ANNs. Specifically, we employ the visualization approach proposed in the previous chapter to the following two tasks:

- **T1**: Exploring the relationships between alternative representations of observations *learned* by ANNs.
- **T2**: Exploring the relationships between artificial neurons.

Although we focus on multilayer perceptrons and convolutional neural networks, our approach is extensible to other types of networks (*e.g.*, LSTM or Elman networks [58]).

The projection-based visualization approach that we propose for **T1** is (sparsely) found in the machine learning literature. However, such projections are typically used for illustrative purposes. In contrast, we show how projections can aid existing approaches for understanding and improving ANNs (Sec. 5.4). Specifically, using three widely studied benchmark image classification datasets, we show how our visualization approach is able to confirm facts that are already known about ANNs, and reveal previously unseen relationships between learned representations. In this context, we also propose a novel visualization of the *evolution* of such learned representations (Sec. 5.4.4).

Our approach towards **T2** is completely new in the context of ANNs, although it is related to techniques developed for feature-space exploration discussed in the previous chapter (see also Sec. 5.2). Similarly to our approach for **T1**, we use projections to represent *similarities* between artificial neurons (given a particular set of input observations).

This chapter is based on the following publication:
Paulo E. Rauber, Samuel G. Fadel, Alexandre X. Falcão, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of the Visual Analytics Science and Technology 2016)*, 23(01), January 2017.

We also propose a novel visualization of the relationships between artificial neurons and classes (Sec. 5.5.2). Although being presented separately, our visualization approaches for **T1** and **T2** should be seen as complementary for understanding ANNs, as we exemplify in Sec. 5.5.2.

This chapter is organized as follows. Section 5.1 briefly reviews our notation and definitions. Section 5.2 relates our work to previous work in machine learning, information visualization, and visual analytics. Section 5.3 details the protocol followed by our experiments. Section 5.4 presents the results of our projection-based visualizations of the relationships between learned representations for different datasets (**T1**), highlighting valuable insights gained from visualization. Section 5.5 presents our projection-based visualizations of relationships between artificial neurons (**T2**). Section 5.6 discusses the limitations of our work. Section 5.7 summarizes our findings and suggests future work.

5.1 Preliminaries

The following is a concise review of definitions introduced in Ch. 2 (specially Sec. 2.3.5).

A dataset $\mathcal{D} = (\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$ is a sequence where $\mathbf{x}_i \in \mathbb{R}^D$ is an *observation*, and $\mathbf{y}_i \in \{0, 1\}^C$ is a *target class assignment*. If $y_{i,c} = 1$, observation $\mathbf{x}_i$ belongs to class c. In this chapter, each observation corresponds to a raw 2D image (flattened into a real vector, as described in Sec. 2.3.5) and belongs to a single class, although these are not limitations of our proposal.

We consider two kinds of ANNs: *multilayer perceptrons* (MLPs) and *convolutional neural networks* (CNNs). Such networks represent parameterized functions $f : \mathbb{R}^D \to (0, 1)^C$, which usually attempt to generalize class assignments from the examples in $\mathcal{D}$. Computation in these networks is performed by *artificial neurons*, which are typically organized into *layers*, as detailed next.

Multilayer perceptrons: In these models, the weighted input to neuron j in layer l is defined as $z_j^{(l)} = b_j^{(l)} + \sum_k w_{j,k}^{(l)} a_k^{(l-1)}$, where $w_{j,k}^{(l)}, b_j^{(l)} \in \mathbb{R}$ are free parameters, and $a_k^{(l-1)}$ is the activation (output) of neuron k in layer $l - 1$ (Fig. 5.1). In other words, each neuron computes a linear combination, plus a bias, of neuron activations from the previous layer. The activation $a_j^{(l)}$ depends on the *activation function* chosen for layer l [12]. In a sigmoid layer, $a_j^{(l)} = 1/(1 + \exp(-z_j^{(l)}))$; in a rectified linear layer, $a_j^{(l)} = \max(0, z_j^{(l)})$; in a softmax layer, $a_j^{(l)} = \exp(z_j^{(l)}) / \sum_k \exp(z_k^{(l)})$.

The *activation of layer* l is defined as $\mathbf{a}^{(l)} = \left(a_1^{(l)}, \ldots, a_{N^{(l)}}^{(l)}\right)$, where $N^{(l)}$ is the number of neurons in layer l. Thus, if we let f denote the function computed by the network and L denote its number of layers (including the input), we have $f(\mathbf{x}) = \mathbf{a}^{(L)}$ when $\mathbf{a}^{(1)} = \mathbf{x}$. Any layer between the first and the last is called a *hidden* layer. A network with more than one hidden layer is called a *deep* neural network [11].

The activation of layer $l > 1$ can be seen as an alternative (learned) representation of the input observation, since the activation of layer l depends only on *learned* parameters and the activation of layer $l - 1$ (see Fig. 5.1). This fact is crucial to our approach. The activations of a given layer for a set of observations (network inputs) is the focus of our visualization.

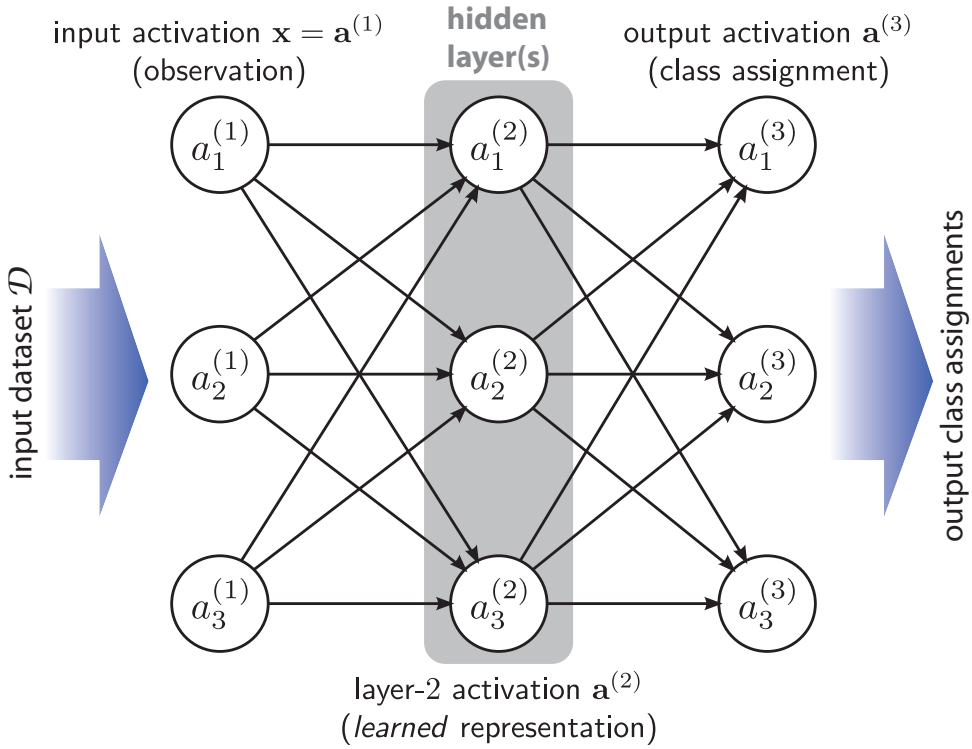


Figure 5.1: Schema of MLP with three layers and three neurons per layer.

Convolutional neural networks: These models typically consist of at least three types of layers: convolutional, (max-)pooling, and fully connected. A *convolutional* layer receives as input a $w \times h$ image with c color channels, and connects each of its neurons to a small window (all channels included) of the input. Neurons compute their weighted inputs and activations as usual. However, each neuron is replicated (with *parameter sharing*) for many input windows, given a pre-defined stride. When the output of all corresponding replicas are organized into a single-channel image, the operation is analogous to multichannel image convolution [84, 112]. The output of a convolutional layer is obtained by stacking the outputs of sets of replicated neurons into a single multichannel image. The number of output channels can be seen as the number of convolutional filters. A *max-pooling* layer reduces the spatial dimensions of a multi-channel image by keeping the highest-value activation in a neighborhood (independently for each channel, for a pre-defined stride), and also outputs a multichannel image. A *fully connected* layer is analogous to an MLP layer, and is only followed by other fully connected layers. The activation of such a layer can also be seen as a learned representation of the input.

Network training: The weights and biases $\boldsymbol{\theta}$ of an ANN are adapted to minimize a cost function J that penalizes prediction errors on the training set $\mathcal{D}$. For example, a softmax output layer is typically combined with the (average) negative log-likelihood cost function J given by

$$J = -\frac{1}{N} \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \sum_{k=1}^{C} y_k \ln a_k^{(L)}, \tag{5.1}$$

where $a_k^{(L)}$ is the activation of neuron k on the last layer L when the network receives $\mathbf{x}$ as input, and N is the number of observations[11]. Note that $-y_k \ln a_k^{(L)} \to \infty$ when $y_k = 1$ and $a_k^{(L)} \to 0$, which characterizes a prediction error.

The process of minimizing J with respect to $\boldsymbol{\theta}$ is called *training*. As J is differentiable with respect to every network parameter, minimization can be attempted by gradient descent. This technique iteratively updates $\boldsymbol{\theta}$ by the rule $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J$, where η is the *learning rate*. In our work, we use (momentum-based) mini-batch stochastic gradient descent [12]. This technique partitions the dataset $\mathcal{D}$ into batches, and approximates $\nabla_{\boldsymbol{\theta}} J$ by using a single batch for each parameter update. After each batch is considered, training has completed one *epoch*. The *hyperparameters* (batch size, learning rate, number of layers, number of neurons per layer, etc) are not affected by training, and are usually chosen using previous experience and cross-validation.

Dimensionality reduction will be employed to create a projection $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$, with $\mathbf{p}_i \in \mathbb{R}^2$, from a layer-$l$ activation set $\mathcal{A} = \mathbf{a}_1^{(l)}, \ldots, \mathbf{a}_N^{(l)}$, where each $\mathbf{a}_i^{(l)} \in \mathbb{R}^{N^{(l)}}$ is the layer-l activation vector corresponding to input observation $\mathbf{x}_i \in \mathbb{R}^D$ (for a given ANN). Such a projection $\mathcal{P}$ attempts to preserve the high-dimensional structure of the activation set $\mathcal{A}$, which is composed of learned representations.

5.2 Related work

Machine learning experts have developed many strategies to design and improve ANNs, since the success of these models is highly impacted by the choice of preprocessing steps and several (interacting) hyperparameters. During training, a common approach is comparing model accuracy on a validation set with accuracy on a training set [119]. This helps diagnosing overfitting (low validation accuracy when compared to training accuracy) and underfitting (low accuracy in both cases).

Manual choice of hyperparameters requires significant expertise and effort, and comprehensive guides have been written on the subject [113, 12]. The high dimensionality of the data and large number of parameters make ANNs hard to interpret, and make improving models a challenging task. Although automatic hyperparameter search is possible [136], it is generally (computationally) expensive.

Visual analytics and information visualization systems have been developed to inspect ANNs, since visual feedback is considered highly valuable by practitioners. For instance, Zeiler and Fergus [159] show how insight gained from visualizing ANNs has enabled them to outperform the state-of-the-art (at the time) on an major image classification benchmark. Their work aims to reconstruct an input image (observation) given a particular output channel of a convolutional layer (also called a *feature map*).

Reconstruction from activations is also investigated by Mahendran and Vedaldi [99]. Erhan *et al.* [39] search, through optimization, inputs that cause high activations in particular neurons, with the goal of understanding their roles.

Yosinski *et al.* [157] visualize feature maps from CNNs trained for image recognition

The rectified linear activation function is not differentiable at 0, but that is usually irrelevant in practice.

as they receive an input video stream, which enables the visual search for filters that detect a particular object. They also extend the work of Erhan *et al.* [39], showing how regularization techniques can be used to generate more interpretable images that cause high activations in a neuron. As will become clear, our approach is *complementary* to these visualizations.

Dimensionality reduction has been previously applied to ANN visualization, due to its scalability in number of dimensions and observations. For instance, Erhan *et al.* [38] use projections of *learning trajectories* to study the effects of unsupervised pre-training. Each point in such a trajectory corresponds to the concatenation of output layer activations for a whole dataset at a given training stage. Closer to our work, projections of hidden layer activations, the subject of Sec. 5.4, have been used as illustrative evidence of model efficacy [36, 105, 137, 106, 63]. Aubry and Russell [6] visualize hidden layer activations using PCA, aiming to understand their invariance with respect to several factors present in real and synthetic images.

In contrast to these works, our work is the first to present a detailed analysis of the insights on classification systems obtainable by projections of hidden layer activations.

Separately, in Sec. 5.5, we use projections to explore relationships between neurons in a hidden layer. This visualization approach is completely new in the context of ANNs, but related to previous work on feature space exploration that we already discussed in the previous chapter [158, 144]. However, in contrast to such previous work, which explores relationships between *input* features (dimensions) to a pattern classification technique, we visualize relationships between features (neuron activations) *learned* by such a technique.

5.3 Experimental protocol

This section details the protocol followed by the experiments that evaluate our visualization approach, which is based on hidden layer activations extracted from a network trained for a given dataset. Our approach is divided into two parts: creating projections from these activations (**T1**, Sec. 5.4), and depicting the relationships between the neurons that originate these activations (**T2**, Sec. 5.5).

Datasets include three well-known image classification benchmarks: MNIST [86], SVHN [110] and CIFAR-10 [83]. MNIST has 50×10^3 training images, 10×10^3 validation images, and 10×10^3 test images (28×28 grayscale images of handwritten digits). SVHN has 63.2×10^3 training images, 10×10^3 validation images, and 26×10^3 test images (32×32 color images of house number digits). CIFAR-10 has 30×10^3 training images, 10×10^3 validation images and 10×10^3 test images (32×32 color photographs in ten object classes). Although the images in SVHN and CIFAR-10 are quite small, which allows fast experimentation, these are not *toy* datasets, and are widely used to evaluate state-of-the-art ANNs [152, 87].

Neural networks of two types are considered, as follows:

1. Multilayer perceptron (MLP): 3072 (784, for MNIST) input neurons, followed by four rectified linear hidden layers of 1000 neurons each. The output layer is softmax with 10 neurons. Dropout [137] is applied from the first hidden layer, growing from 0.2 to 0.5 in steps of 0.1 per layer.

2. Convolutional neural network (CNN): $32 \times 32 \times 3$ input image ($28 \times 28 \times 1$, for MNIST), followed by a convolutional layer with 32 $3 \times 3 \times 3$ (or $3 \times 3 \times 1$) filters, a convolutional layer with 32 $3 \times 3 \times 32$ filters, a 2×2 max-pooling layer (dropout 0.25), a convolutional layer with 64 $3 \times 3 \times 32$ filters, a convolutional layer with 64 $3 \times 3 \times 64$ filters, a 2×2 max-pooling layer (dropout 0.25), a fully connected layer with 4096 (or 3136) neurons (dropout 0.5), a fully connected layer with 512 neurons, and a softmax output layer with 10 neurons. All convolutional and fully connected layers (except the output) are rectified linear.

While larger models (in number of layers and parameters) are used for certain difficult classification tasks, the architectures sketched above are fully realistic, typical for image classification tasks, and sufficiently complex to warrant exploration.

Training is performed by momentum-based mini-batch stochastic gradient descent [12]. For MLPs, the batch size is 16, learning rate is 0.01, momentum coefficient is 0.9, and learning decay is 10^{-9}. For CNNs, the batch size is 32, learning rate is 0.01, momentum coefficient is 0.9, and learning decay is 10^{-6}. Initial weights for a neuron in layer l are sampled from an uniform distribution on $[-s, s]$, where $s = [6/(N^{(l-1)} + N^{(l+1)})]^{1/2}$, and biases start at 0. We manually chose these hyperparameters, together with the aforementioned architectures, based on cross-validation using the pre-defined validation sets. After the hyperparameters were chosen, we trained the models again using all data except the pre-defined test sets.

Table 5.1 summarizes the test set accuracy (AC, fraction of correctly classified observations) of our networks, and compares it to state-of-the-art networks, some of which also use preprocessing and data augmentation. Clearly, our networks achieve good accuracy on benchmark datasets. As such, they should be seen as realistic from an application perspective.

Table 5.1: Test set accuracies for our two architectures and three datasets.

Dataset/Model	MLP	CNN	State of the art
MNIST	98.52%	99.62%	99.79% [152]
SVHN	77.38%	93.76%	98.08% [87]
CIFAR-10	52.91%	79.19%	91.78% [87]

Activations for a given layer, the subject of our analysis, are extracted for a random subset of 2000 observations from the test sets, strictly to facilitate visual presentation. This subset is always the same for a given dataset. In two cases, we also extract activations from a random subset of a training set (Sec. 5.4.2). For CNNs, we only extract activations from fully connected layers.

Projections are created using a fast (approximate) implementation of t-distributed stochastic neighbor embedding (t-SNE, Sec. 2.6.4) [146], using default recommended parameters. We chose this technique based on its widespread popularity, and proven ability to preserve neighborhoods and clusters in projections [148].

As in the previous chapter, we visualize projections as scatterplots, with points colored to show class assignment. We measure projection quality by the neighborhood hit (NH), which indicates how well classes are visually separated [116]. For a given k (in our work,

$k = 6$), the NH for a point $\mathbf{p}_i$ is the ratio of its k-nearest neighbors that belong to the same class as its corresponding observation $\mathbf{x}_i$. The NH for a whole projection is the average NH over its points. When displaying classification results for a test set in a projection, we use triangle glyphs to show misclassified observations (points), colored by their (incorrect) classifications (*e.g.*, inset in Fig. 5.3b).

Implementation of all our work is based on Python, Keras, Theano [9], NumPy [149], and scikit-learn [118]. Our visual exploration was conducted using the feature space exploration tool presented in the previous chapter.

5.4 T1: relationships between activations

This section presents the results of the experiments conducted to evaluate our proposed visualization of relationships between learned representations of observations (**T1**). For brevity, we focus on the most distinctive results and insights obtained for each dataset (Secs. 5.4.1 - 5.4.3). In Sec. 5.4.4, we present a novel visualization of the *evolution* of learned representations.

5.4.1 MNIST: exploring effects of training

The MNIST dataset is known as a relatively easy classification benchmark. This is confirmed by the clear visual separation between classes in the (raw data) projection of a subset of 2000 test observations (784-dimensional vectors), shown in Fig. 5.2. Points are colored according to their classes.

What untrained ANNs see: As already mentioned, we aim to understand the relationships between the alternative representations *learned* by ANNs trained for pattern classification. Firstly, consider an untrained MLP, whose parameters are randomly initialized according to Sec. 5.1. It is reasonable to hypothesize that a projection of the hidden layer activations of this MLP would have a significantly poorer visual separation between classes than the one shown in the projection in Fig. 5.2.

In Fig. 5.3a, we show the projection of the last MLP hidden layer activations *before* training, for the same test subset used in Fig. 5.2. The hypothesis outlined in the previous paragraph was contradicted, since both projections (Figs. 5.2 and 5.3a) show similar visual separation between classes. The good separation in Fig. 5.3a cannot be due to class information, since class labels are not used by the dimensionality reduction technique. Thus, there must be a clear structure in the hidden layer representations *before training*, which leads to the reasonably good NH of 83.78%. We are unaware of previous visualizations showing this qualitative insight, which could be used to compare different ANN initialization strategies. While it would be possible to use the hidden layer activations to train a separate learning algorithm and evaluate its accuracy on a test set, such approach would be more time-consuming, and would not convey the *structure* of the data in an easily interpretable manner. For instance, see the relationship between the visual clusters that correspond to (visually similar) classes 4 and 9 in Fig. 5.3a.

Available in `http://www.cs.rug.nl/svcg/People/PauloEduardoRauber-featured`.

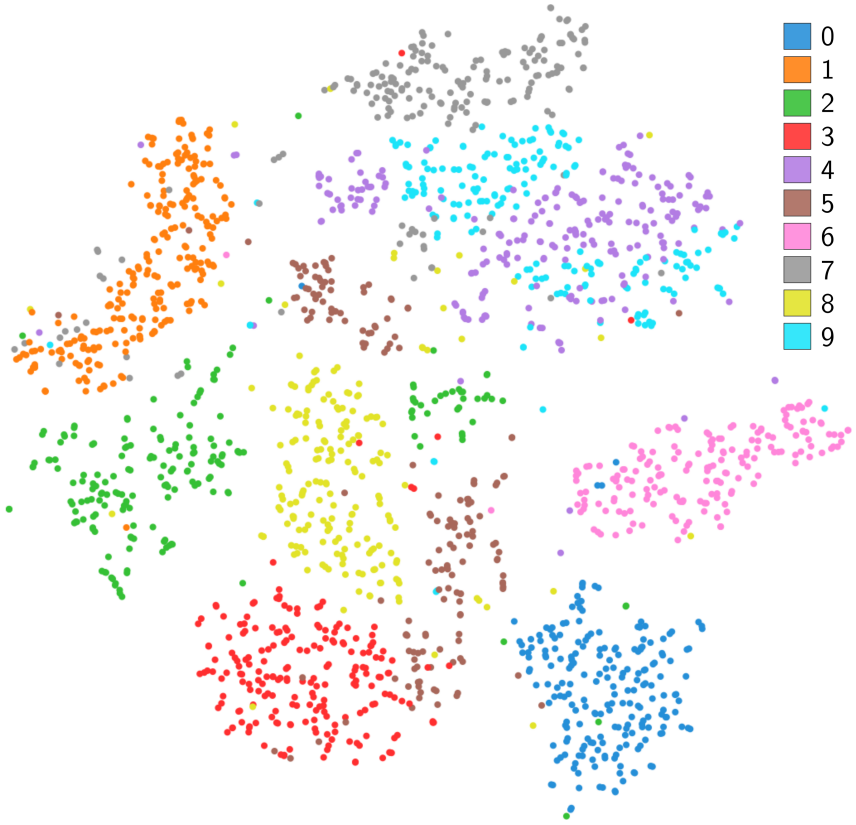


Figure 5.2: Projection of observations, *MNIST* test subset (NH: 89.12%).

Training effects: A second natural hypothesis is that visual separation between classes would become better *after* training. This is related to the commonly-held view that ANNs learn to detect higher-level features that are useful for class discrimination [11]. To study this hypothesis, consider the projection of the last MLP hidden layer activations *after* 100 training epochs (Fig. 5.3b). Compared to Figs. 5.2 and 5.3a, we see a dramatic improvement in the visual separation between classes. Hence, the learning process definitely arrived at an alternative representation of the data that captures class structure, which is reflected by the projection.

Understanding misclassifications: Figure 5.3b shows several *visual outliers*, *i.e.*, points whose neighbors belong to a different class. Assuming the projection preserves the high-dimensional data structure, we could suspect that such outliers would be misclassified by the ANN. To check this, we color all points by their classes, and mark misclassifications by triangle glyphs. The inset in Fig. 5.3b shows several such misclassifications. When inspected, the visual outliers often correspond to observations that even humans would recognize as hard cases. For instance, Fig. 5.3b shows how an image of the digit 3 is (understandably) mistaken for the digit 5, and placed near the visual cluster corresponding to digit 5. This example shows that, despite the fact that projections may sometimes not fully preserve the data structure, as we discuss in Sec. 5.6, they are often predictive about class assignment. In other words, the similarities between hidden layer activations (shown by the projection) are a good predictor of the final class assignment by the ANN. This type of feedback is particularly useful when projections are combined with

background knowledge and manual inspection of ANN inputs, as we continue to show in the next section.

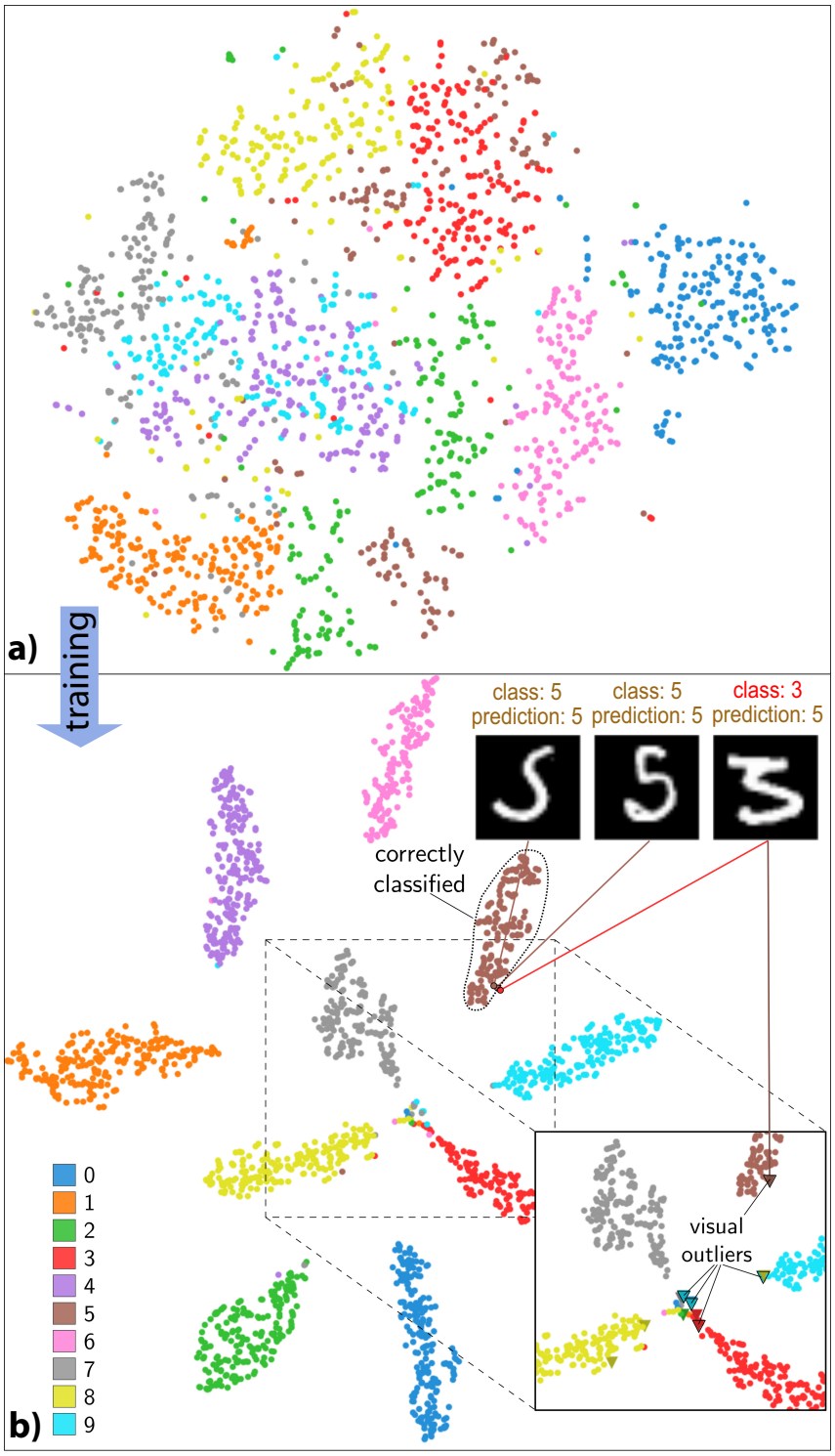


Figure 5.3: Projection of the last MLP hidden layer activations, MNIST test subset. a) Before training (NH: 83.78%). b) After training (NH: 98.36%, AC: 99.15%). Inset shows classification of visual outliers.

5.4.2 SVHN: interpreting visual clusters

This section presents a compelling case for the visualization of learned representations in a second dataset (SVHN). Visualization provides a particular qualitative insight that is not easily available by other means.

The SVHN dataset is much more challenging for classification than MNIST. This is reflected, before training, in the visual separation between classes in the projection of the last hidden layer activations of an MLP, which is considerably poorer for SVHN (Fig. 5.4) than for MNIST (Fig. 5.3a). This is also confirmed by the corresponding NHs. Just as for MNIST, the visual separation is significantly improved after training, as shown by Fig. 5.5b.

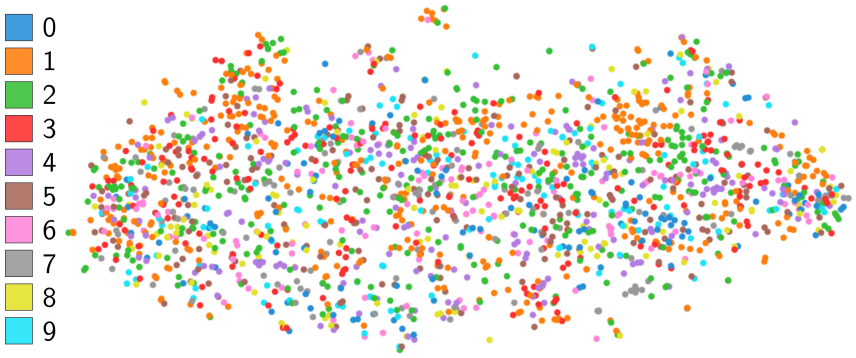


Figure 5.4: Projection of the last MLP hidden layer activations before training, *SVHN* test subset (NH: 20.94%). Poor class separation is visible.

Comparing different layers: The projections presented so far showed activations of last MLP hidden layers. However, our MLP architecture has four hidden layers. It is often hypothesized (but not usually supported by evidence) that a properly trained deep ANN has activations at later layers that correspond to discriminative higher-level features of the original observations [11]. We can verify this by inspecting activations of earlier layers which, in our case, have the same number of neurons. Consider the projection of the activations of the *first* MLP hidden layer after training (Fig. 5.5a). Visual separation between classes is clearly inferior to the one shown in the last hidden layer (Fig. 5.5b). We saw this phenomenon for most of the ANNs trained in our study. This is a new finding, which is not documented in the existing literature on ANNs. Note that there are no easy alternatives to obtain such an insight. For instance, confusion matrices could be used to diagnose the confusion between classes for a learning algorithm trained on the hidden layer activations. However, a confusion matrix would not convey nearly as much information about the structure of the data as a projection. Furthermore, a confusion matrix for a 10-class problem (our case) has 45 independent scalar values (after considering symmetries), which makes it quite difficult to inspect.

In comparison to the results obtained with the MLP (AC: 77.38%), the CNN obtains considerably better classification results on the test set (AC: 93.76%). Figure 5.6 shows the projection of the last CNN hidden layer activations after training. Clearly, visual separation and classification results improved together (cf. Fig. 5.5b), which is another example of the predictive power of projections.

Improvement based on visual feedback: We now present a particularly salient example of the value of the insight provided by projections. In Fig. 5.5b and (most notably) in Fig. 5.6, we notice a very distinctive pattern: each class (color) seems to be split into two visual clusters. Upon further inspection (brushing points), we found that one of the same-colored visual clusters corresponds to dark digits on light backgrounds, and the other to light digits on dark backgrounds (see examples in Fig. 5.6). We are unaware of previous work that documents such a remarkable pattern (visual or otherwise) in learned representations for the SVHN dataset, even though this dataset has been extensively used to evaluate ANNs in hundreds of publications. This is an example of how qualitative feedback can be hard to obtain by the typical quantitative approaches used to evaluate ANNs.

Since the projection in Fig. 5.6 suggests that, for each class, there are two kinds of images that have dissimilar internal representations, we naturally suppose that removing such (apparently unnecessary) variability in the input images would improve classification efficacy. To evaluate this, we preprocessed the images in SVHN in a simple manner: we apply the Sobel operator, after a small Gaussian blur, to approximate the gradient magnitude of the grayscale counterpart to each image. This yields grayscale images that are bright on the edges between background and foreground, and avoids the task of detecting if a digit is light or dark, which is not trivial given the high variability of the images. Next, we use the experimental protocol in Sec. 5.3 to classify the preprocessed test set. We obtain an increase in accuracy of 3.96% (1030 test observations) with the MLP, and 0.65% (169 test observations) with the CNN. While the CNN gain is small, we stress that it was obtained by *exactly* the same network architecture that was used for the original images. In contrast, the MLP gain is quite significant. The difference in gains for the two architectures can be explained by two facts. Firstly, it is easier to obtain gains when a model is further away from ideal performance, as in the MLP case. Secondly, our preprocessing is highly related to the operation of convolutional layers. This can be sufficient to enable good generalization in the CNN case, given the large amount of labeled data available for training. Finally, we also note that, in contrast to Figs. 5.6 and 5.5b, the corresponding projections of the preprocessed test subset (omitted for brevity) do not show two distinct visual clusters for each class. The increase in neighborhood hit (MLP 9.06%, CNN 1.43%) for those projections also mimics the increase in accuracy. Overall, these facts corroborate our hypothesis about the *semantics* of the internal ANN representations.

We note that it is already known in the literature that preprocessing this dataset (by local contrast normalization) sometimes leads to better performance [54]. However, this procedure was never justified by the foreground-background insight that we discovered. In the general case, a practitioner might be unaware of important preprocessing steps for a particular domain or dataset. In such situations, we claim that projections can provide highly valuable qualitative information about the representations learned by ANNs. As we already argued, such feedback is very hard to obtain by other (non-visual) means.

Explaining misclassifications: Consider the cyan point outlined in Fig. 5.6, which corresponds to digit 9, but is placed near the green cluster corresponding to dark digits 2 on light backgrounds. Notice how the dark border to the right of the digit 9 could

be interpreted as a malformed digit 2. Knowing the semantics behind the green cluster, we can explain the misclassification more easily. While an experienced practitioner may have guessed why the misclassification occurred without the visual feedback, the *semantics* assigned to the visual clusters (found through visualization) provides extra evidence about the misclassification. In the general case, misclassification causes may be less obvious, and the visualization of learned representations becomes even more useful. Understanding the causes of misclassifications is useful both for improving models and for deciding when a model has achieved satisfactory performance.

Assessing training: All the projections presented so far were created from activations from a subset of a *test* set. However, insight can also be gained by inspecting projections of a subset of a *training* set. For training data, it is natural to expect that the visual separation between classes will be even better than in test data. To verify this, we compare the projection of a subset of the SVHN training set activations in the last MLP hidden layer (Fig. 5.7) with that of the corresponding test set (Fig. 5.5b). Indeed, we see a better visual class separation in the former, which is also reflected by a better NH (71.43% *vs* 67%). Comparing the two visual separations (training *vs* test data) supports several assessments. Firstly, a badly separated training set projection may indicate a poorly trained network, which has low chances of performing well on test data. A very well separated training set projection and a poorly separated test set projection may indicate poor generalization (caused, for example, by overfitting). Such assessments can also be restricted to *parts* of a projection. In Fig. 5.7, for example, we see bad visual separation in the center. This is also the area where most classification errors (marked by triangles) are found.

In the CNN case, similar results are obtained by comparing Fig. 5.8 (projection of last CNN hidden layer activations after training, for a training subset) with Fig. 5.6 (corresponding projection for the test subset). Comparing projections from different architectures is also insightful: in our case, we see that the CNN performs considerably better on the training set than the MLP, which matches the perceived visual separation and NH for Figs. 5.7 and 5.8. In fact, the CNN yields only two training set misclassifications. By brushing them in the projection (Fig. 5.8), we discover that one of them is incorrectly labeled (bottom right inset in Fig. 5.8).

Discovering potential overfitting: Figure 5.8 provides a final interesting insight. Consider the two gray points placed near the orange visual cluster (top left). Although the CNN assigns the correct class to these points (digit 7), the representation of the last hidden layer places them near orange points (digit 1). This appears to be due to the fact that the two images of the digit 7 actually resemble images of the digit 1. The placement of these two points in the projection can be a sign that the last layer learned to *work around* (overfit) the internal representation (recall that we are looking at training data). This could mean that the network will not work well in similar cases, classifying digits 7 as digits 1. Naturally, care must be taken when drawing conclusions from the placement of small sets of points, as we discuss in Sec. 5.6.

5.4.3 CIFAR-10: interpreting confusion zones

The CIFAR-10 dataset is considerably more challenging for classification than the previous two (Tab. 5.1). This dataset provides another example where poor visual separation between classes predicts low classification accuracy. The projection of the last CNN hidden layer activations after training shows significant overlap between visual clusters (Fig. 5.9), matching the somewhat low classification accuracy (78.7%). Similarly to Fig. 5.7, the area with poorest separation between classes, or *confusion zone*, predicts well where most misclassifications occur (triangles in the center of Fig. 5.9).

As in Sec. 5.4.2, inspecting the visual outliers is also interesting. Consider the outlier in the middle of the large cyan visual cluster in Fig. 5.9. Since the class in cyan corresponds to truck images, and the outlier observation (automobile) looks very similar to members of that class, it is not surprising that the corresponding point becomes a visual outlier given the learned representation. Many other examples like this can be found in the projection.

5.4.4 Evolution of learned representations

The previous sections presented projections of learned representations (activations) for combinations of datasets, training stages (epochs), and layers. However, a single projection does not show how these representations *evolve*. The goal of this section is to introduce a compact visualization that summarizes two dimensions of evolution: inter-epoch and inter-layer. Given an observation and a layer, inter-epoch evolution refers to the changes to the activations of that layer that are consequence of learning (parameter changes as epochs progress). Given an observation and an epoch, inter-layer evolution refers to the changes in internal representation as the observation "flows" through the layers of the network.

Let $\mathcal{A}[1], \ldots, \mathcal{A}[T]$ be a sequence of sets of (high-dimensional) activations, where each activation $\mathbf{a}[t] \in \mathcal{A}[t]$ originates from the same observation as a single activation $\mathbf{a}[t+1] \in \mathcal{A}[t+1]$. One way to visualize the evolution in such sets of activations is dimensionality reduction, applied in such a way that changes in the resulting projections will reflect changes in the corresponding high-dimensional data. This can be done by creating a projection $\mathcal{P}[t]$ for each activation set $\mathcal{A}[t]$. When doing this, it is essential to eliminate variability between projections that does not reflect changes in the high-dimensional data. The (arguably) simplest way to do this is to compute $\mathcal{P}[t]$ independently for each t, and use point-cloud registration [55] to align the resulting projections. However, dimensionality reduction techniques, including t-SNE, often yield large changes in *global* visual cluster placement for *small* data changes, which registration cannot eliminate [49]. For iterative techniques such as t-SNE, another intuitive strategy is to initialize the positioning in $\mathcal{P}[t+1]$ with the previously computed $\mathcal{P}[t]$. We verified that this is a poor alternative, as it significantly biases the sequence of projections to show the evolution due to initialization in a (likely) *better* state with respect to the optimization goal.

In this section, we employed a simple strategy: computing a (randomly initialized) projection $\mathcal{P}$ of $\mathcal{A}[1]$ using t-SNE, and using $\mathcal{P}$ to initialize each $\mathcal{P}[t]$. In Chapter 6, we propose a new technique to overcome the issues with this and the previous strategies, which may be combined with the visualization approach that we propose next.

The resulting sequence of projections can be visualized in several ways. We discard animation, since it is hard to track the motion of a large number of colored points over many frames. An alternative is to create a 2D trail for each sequence $\mathbf{p}_i[1], \ldots, \mathbf{p}_i[T]$ of corresponding points. However, directly drawing these trails causes a large amount of clutter and occlusion. We address this by using trail bundling. This is analogous to earlier applications of bundling to visualize vehicle and eye-tracking trails [71]. We employ a recent high-performance GPU-based bundling algorithm [151], which allows a high degree of control in real time. The following examples show how the resulting bundled images can be explored.

Inter-layer evolution: Figure 5.10 shows the inter-layer evolution of a MNIST test subset (after training). The bundled image summarizes a sequence of four projections, one per hidden layer, shown as thumbnails. Trail hues encode classes, and edge brightness encodes layer number (depth). Thus, the brightness gradient shows how activation data "flow" through the four network layers. The bundle shapes show that the visual clusters are quite stable over all layers. Hence, the network arrives at a reasonably good separation between classes already at early layers. The gradients also show that some visual clusters become more compact in later layers (*e.g.*, tight bright area in the green cluster, whose evolution is indicated by the gray arrows in the figure), and that some clusters distance themselves from the others (*e.g.*, brightness pattern in the purple cluster). Thus, later network layers *strengthen* the class coherence and separation achieved by the first layer. We also see that there are only a few visual outliers (stray trails) that connect distinct visual clusters. Therefore, only few observations change clusters as the activation data flow through the network. In summary, we infer that the network layers after layer 1 mainly *refine* cluster coherence.

Inter-epoch evolution: We could employ the same ideas to visualize inter-epoch evolution. However, our results (omitted for brevity) in this case show that the images are significantly harder to interpret. This is due to a combination of large changes in the the very first epochs, high intra-visual-cluster variance between epochs, and a much larger number of *frames* (typically hundreds) to be summarized.

For this reason, we employed a different strategy to visualize inter-epoch evolution in this case. Consider again the sequence $\mathcal{A}[1], \ldots, \mathcal{A}[T]$ of activation sets. For inter-epoch evolution, $\mathcal{A}[t] \subset \mathbb{R}^k$ for a fixed k, for all t. Hence, we can create a projection for the set $\bigcup_t \mathcal{A}[t]$, which contains activations for all epochs. As we compute a single projection, there is no spurious inter-frame variation. However, as we will discuss in Chapter 6, this strategy also has significant drawbacks.

Figure 5.11 shows the inter-epoch evolution for the last CNN hidden layer activations using this strategy, from epochs 0 to 100, in steps of 20 (12×10^3 points in total). Hues indicate class, and brighter edge fragments correspond to later epochs. The thumbnails in Fig. 5.11 show points from three selected epochs. It is interesting to note how the dimensionality reduction technique placed the points corresponding to earlier epochs (darker) in the center of the projection, considering that it does not explicitly receive this information. This phenomenon also happens for SVHN and CIFAR-10.

Finally, we note that our choice of bundling algorithm provides a high degree of control

over the level of *trail simplification* [151], which leads to a visualization that can also be explored in different levels of abstraction.

5.5 T2: relationships between neurons

The projections shown in Sec. 5.4 help understanding the relationships between the learned representations of *observations*. However, they do not represent relationships between the *neurons* in a given layer, or how neurons interact to fulfill their discriminative tasks. For this, we complement the *activation projections* shown so far by *neuron projections*. In a neuron projection, each point depicts a neuron. Points are placed in 2D based on the *similarity* between neurons. To our knowledge, this is the first time that artificial neurons are visualized this way.

As in the previous chapter, we define the dissimilarity $d_{i,j}$ between neurons i and j as $d_{i,j} = 1 - |r_{i,j}|$, where $r_{i,j}$ is the (empirical) Pearson correlation coefficient between neurons i and j on a given set of layer-l activations (recall that each element of an activation vector is a neuron output). This metric captures both positive and negative linear correlations between pairs of neurons. From the matrix of pairwise dissimilarities, we compute a projection using (absolute metric) multidimensional scaling (MDS, Sec. 2.6.3) [17]. As we confirmed through preliminary experiments, MDS presents more coherent relationships between neurons that are discriminative for a particular class, which is important for a neuron projection (see Secs. 5.5.1 - 5.5.2).

5.5.1 MNIST dataset

We use the MNIST dataset to introduce neuron projections. Figure 5.12c shows the activation projection and Figure 5.12d the corresponding neuron projection for the last CNN hidden layer activations, after training. Ignoring the colors for now, we see no clear pattern in the neuron projection (Fig. 5.12d), except for some ill-defined visual clusters. We next color each point (neuron) based on its ability to discriminate between class 8 (marked yellow in Fig. 5.12c) and all other classes, computed by a standard feature selection technique, using extremely randomized trees [51] (Sec. 2.4.4). A very clear pattern emerges: all discriminative neurons for class 8 are placed near each other in the neuron projection. In contrast, consider the corresponding activation/neuron projections for the same hidden layer *before* training (Figs. 5.12a,b): the discriminative neurons for class 8 are *scattered* over the neuron projection. This shows that training creates sets of highly related neurons, which work together in the classification task. An analogous phenomenon can be observed for all other classes (omitted for brevity).

We can use feedback about the relationships between neurons and classes to diagnose the absence of specialization for a particular class in a given layer. As a related example, consider *dropout*, a widespread heuristic for training deep ANNs [137]. Dropout is often justified by its hypothesized capacity to inhibit co-adaptation of artificial neurons [137]. We believe our approach could be applied to *qualitatively* investigate and compare this and similar heuristics (*e.g.*, DropConnect [152]), which still are poorly understood [152].

5.5.2 SVHN dataset

Many feature selection methods provide a score that may be employed to measure the importance of a given neuron (feature) to discriminate between a given class and other classes (see Sec. 2.4). We show next how this information can be used to depict how each neuron contributes to class discrimination.

For a given feature scoring technique (extremely randomized trees, in our example), each neuron j receives a normalized score $s_{c,j} \in [0, 1]$, which measures the power of neuron j, relative to other neurons, to discriminate between class c and all other classes. We associate neuron j to the class $c_j^* = \arg\max_{c'} s_{c',j}$. We depict this by coloring point j with the hue associated to class c_j^*, and with a saturation given by $s_{c_j^*,j}$. We call this depiction a *discriminative neuron map*. Notice that the score $s_{c,j}$ is normalized over neurons for each class, so a highly saturated point in the visualization may have a low *absolute* discriminative power.

Figure 5.13 shows the discriminative neuron map for the SVHN test subset, last hidden layer activations, after training. The presence of compact visual clusters shows how the entire set of neurons can be (almost) partitioned into groups with related discriminative roles (specializations), even though the neuron projection is created without any class information.

The activation and neuron projections can be combined to elucidate the role of particular neurons. Consider neuron 460, which is highly associated to class 3 according to Fig. 5.13. The activation of this neuron is encoded using colors in Fig. 5.14, for all inputs in the test subset. According to that image, neuron 460 is responsible for finding one of the two red visual clusters in the projection (see bottom left inset in Fig. 5.14), which corresponds to images of the digit 3 on a light background (as we discovered through visualization in Sec. 5.4.2). It is also interesting to note that an observation that has a high activation for that neuron, and belongs to another visual cluster, resembles a digit 3 upon closer inspection (digit 5 on a light background, top left inset in Fig. 5.14). Obtaining these informations by typical approaches employed in machine learning would be significantly more difficult and time-consuming, which is key to the importance of our visual approach. Finally, as already mentioned in Sec. 5.1, understanding the role of particular neurons in ANNs is considered a very important problem, for which the discriminative neuron map is a novel approach.

5.6 Discussion

In this section, we discuss several important aspects of our proposed visualizations and the experimental analysis conducted to evaluate them.

Scalability: Although dimensionality reduction is among the most scalable methods for high-dimensional data visualization, it still has some issues. Firstly, visual clutter occurs when visualizing a large number of activations or neurons. Secondly, considering the activation projections, although our particular choice of technique (Barnes-Hut t-SNE) is computationally scalable, it still requires approx. 10 minutes to compute a projection containing 70×10^3 50-dimensional observations [146]. Fortunately, preliminary

experiments with dimensionality reduction techniques that are able to deal with hundreds of thousands of observations at interactive paces [116, 75] were also promising. Thirdly, we use categorical color-coding to show class information. This creates well-known clutter and color-distinguishing challenges in scatterplot visualization when the number of classes is large.

Techniques: Our choice of (Barnes-Hut) t-SNE is justified by its widespread popularity and well-known ability to preserve clusters and neighborhoods in projections [148]. Although the latter property is very important to understand relationships between learned representations, our proposal is not highly coupled to t-SNE. Similarly, neuron projections are not coupled to absolute metric MDS. In both cases, other dimensionality reduction techniques can be used, provided that they preserve neighborhoods and distances well, respectively.

Coverage: As an experimental study that involves many free parameters, our conclusions are limited to the datasets and networks that we presented. However, our findings for all datasets and networks were consistent. In particular, there were no cases where projections would not provide any useful feedback. Additionally, the extent of our validation (*i.e.*, experimental protocol, number of datasets) is in line with comparable works in visual analytics and machine learning.

Validity: We employed good practices to train artificial neural networks in well-known benchmark datasets, and carefully detailed our experimental protocol to maximize reproducibility.

It is extremely important to address a specific threat to the validity of our approach: the fact that dimensionality reduction techniques provide few *quality* guarantees, and may introduce misleading visual artifacts [101]. For instance, different initializations of t-SNE may or may not yield the visual outliers presented in Sec. 5.4. To solve this issue, users should primarily evaluate the quality of a given projection using existing metrics, such as those presented in [101, 7], as already mentioned in the previous chapter. Such metrics support both global assessment (overall quality of an entire projection) and local assessment (*i.e.*, whether a subset of points is placed well).

If a projection (or some of its parts) has poor quality, it should be discarded from further use. Conversely, if a projection (or some of its parts) has high quality, the patterns it shows are actually present in the data, an can be relied upon. As a side note, it should be clear that most interesting phenomena observed in the projections in Secs. 5.4 and 5.5 would be extremely unlikely artifacts (*e.g.*, visual cluster separation, partition of visual clusters between light/dark digits on dark/light backgrounds, and partitioning of neurons into specialties). Finally, we note that the feedback given by (activation) projections for classification problems is, in a sense, asymmetric: clear visual separation between classes surely implies an easy classification task, whereas unclear separation does not necessarily imply a difficult task.

5.7 Conclusion

In this chapter, we have shown how dimensionality reduction can be used to visualize the relationships between learned representations (**T1**) and between neurons (**T2**) in artificial neural networks. Concerning the first task, our visualizations support the identification of confusion zones, outliers, and clusters in the internal representations computed by such networks. Separately, we also show how to visually track inter-layer and inter-epoch evolution of learned representations. Concerning the second task, we enable the inspection of relationships between neurons and classes (specialization), and similarity between neurons. In experiments on traditional benchmark datasets, we have shown that both our contributions can provide valuable visual feedback for network designers. This feedback may confirm the known, reveal the unknown, and prompt improvements along the classification pipeline, as we have shown through concrete examples.

There are several possibilities for future work. They include visualizing representations learned by recurrent networks, which currently achieve state-of-the-art results in many sequence-related tasks [58, 57]. The sequential nature of these networks introduces yet another challenge for visualization. Our approach for evolution visualization would also benefit from dimensionality reduction techniques designed specifically for time-dependent datasets. We address precisely this issue in the next chapter.

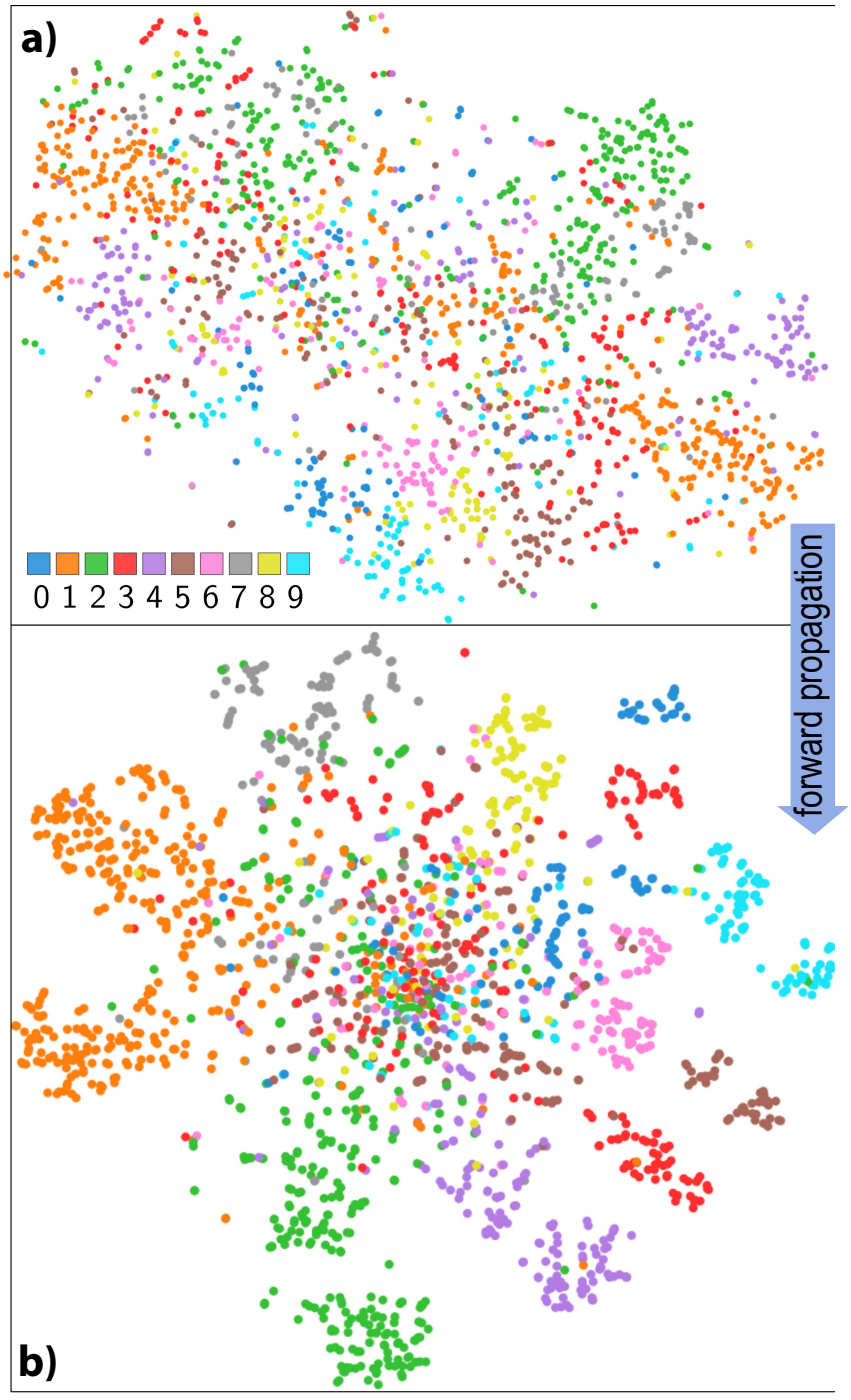


Figure 5.5: Projection of the MLP hidden layer activations after training, SVHN test subset. a) First hidden layer (NH: 52.78%). b) Last hidden layer (NH: 67%).

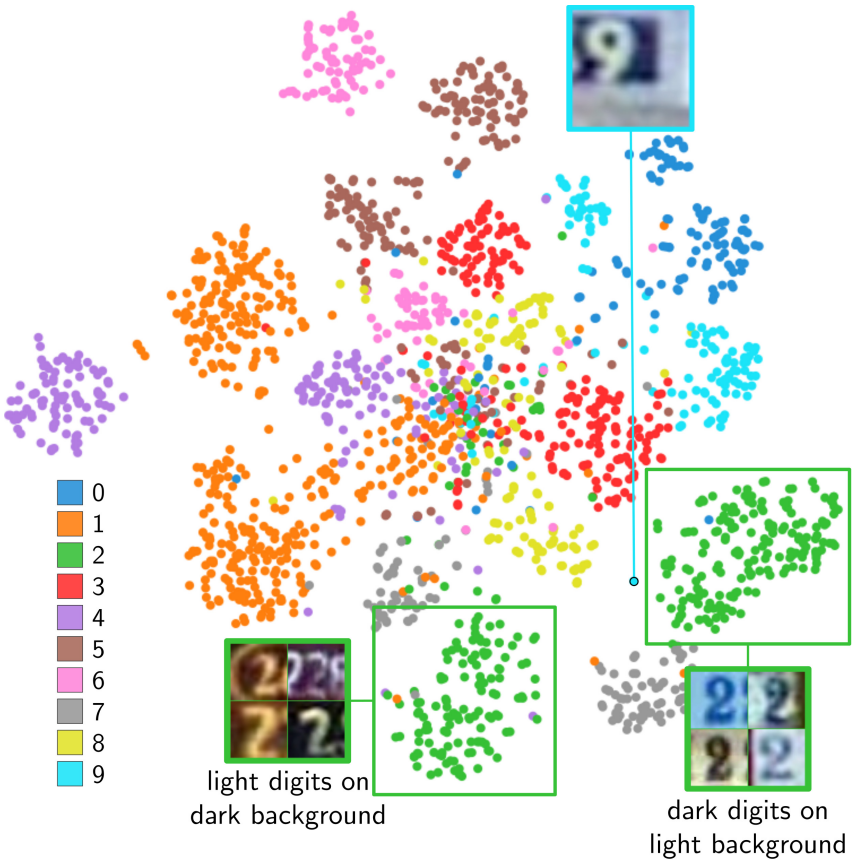


Figure 5.6: Projection of the last CNN hidden layer activations after training, *SVHN* test subset (NH: 85.02%). Insets show example observations (images) from the visual clusters.

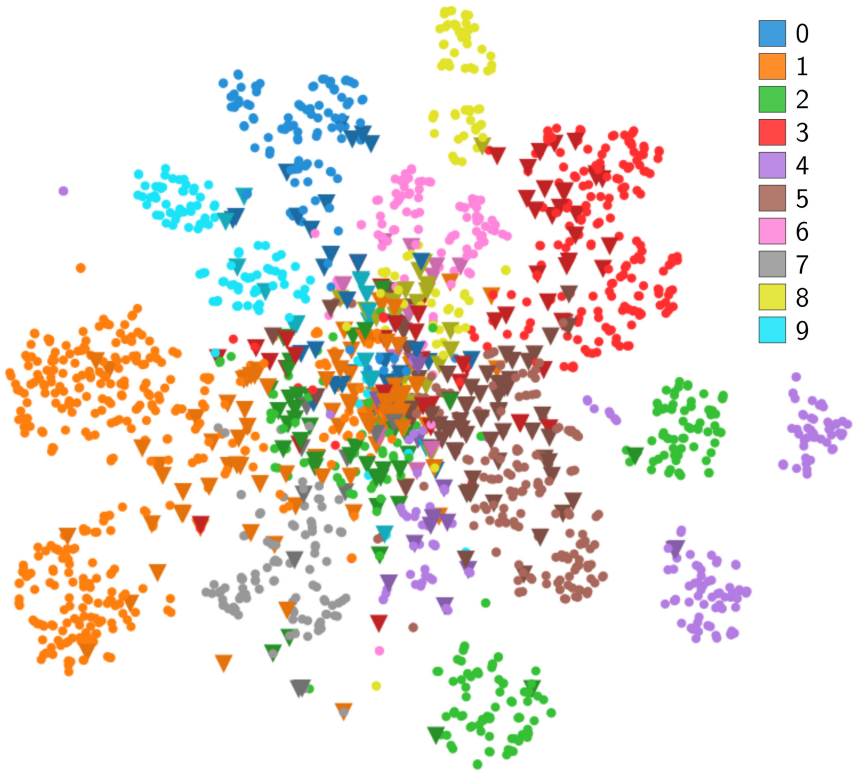


Figure 5.7: Projection of last MLP hidden layer activations after training, *SVHN* training subset (NH: 71.43%, AC: 81.65%).

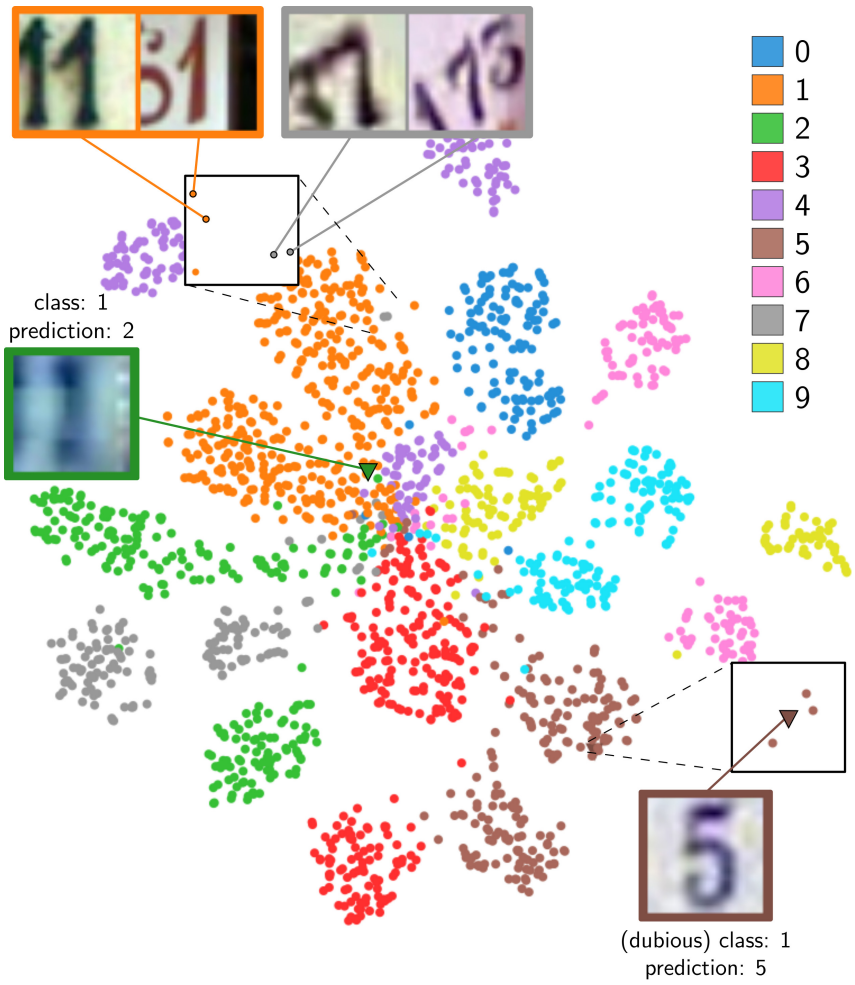


Figure 5.8: Projection of last CNN hidden layer activations after training, *SVHN* training subset (NH: 93.83%, AC: 99.9%).

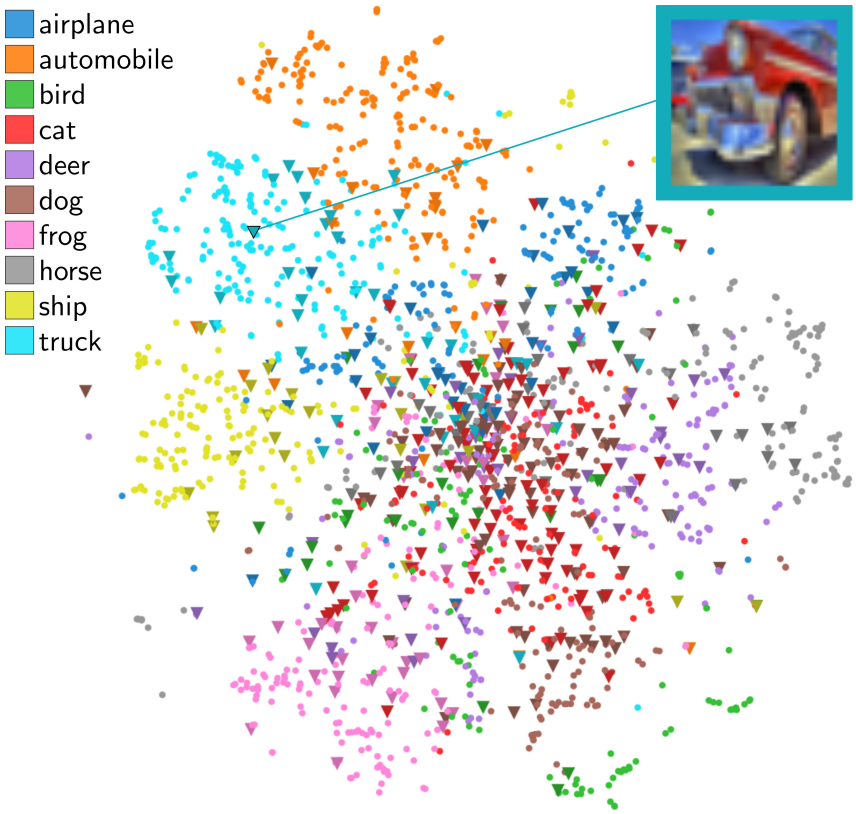


Figure 5.9: Projection of last CNN hidden layer activations after training, *CIFAR*-10 test subset (NH: 53.43%, AC: 78.7%).

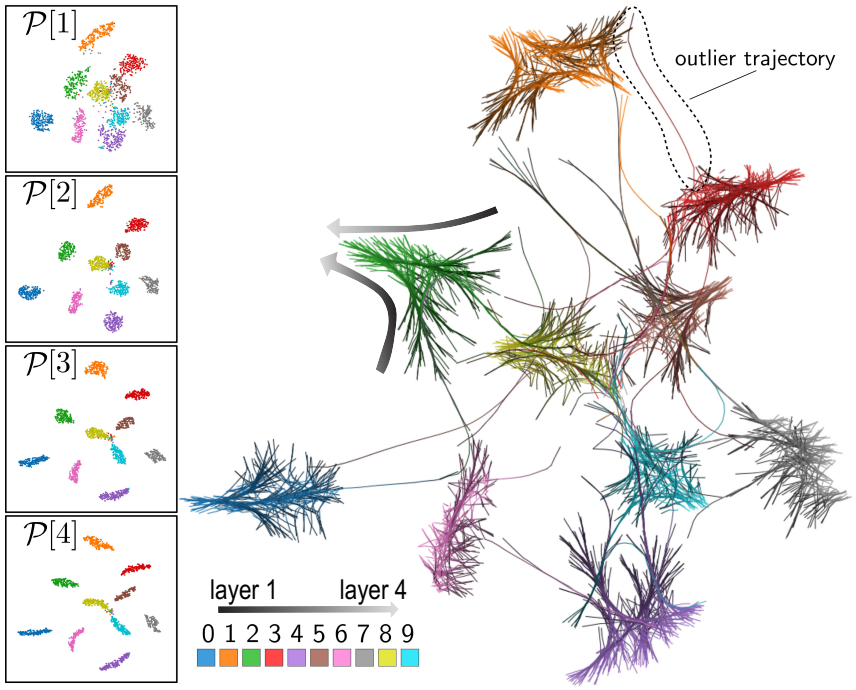


Figure 5.10: Inter-layer evolution, four MLP hidden layers after training, *MNIST* test subset. Brighter trail parts show later layers.

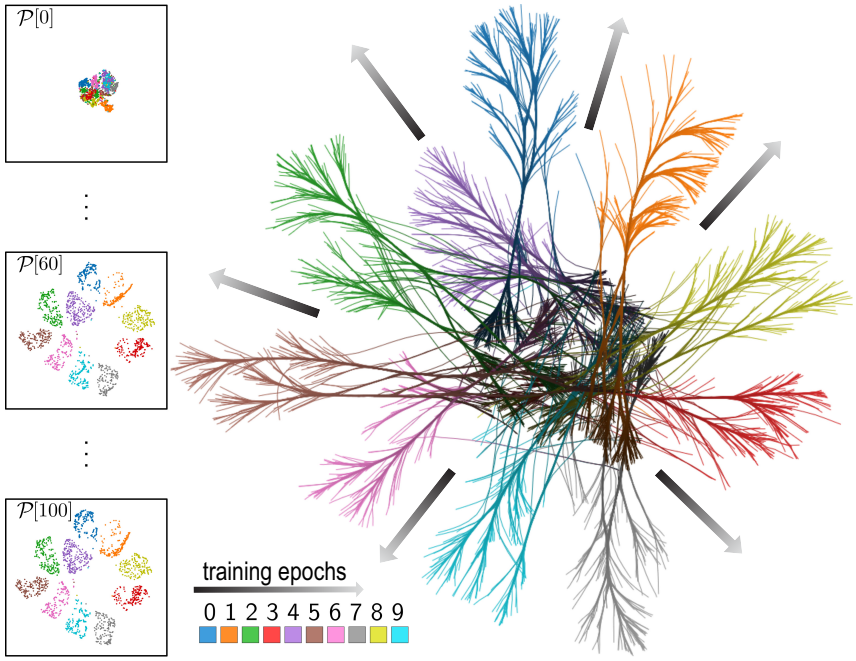


Figure 5.11: Inter-epoch evolution, last CNN hidden layer, epochs 0-100, in steps of 20, *MNIST* test subset. Brighter trail parts show later epochs.

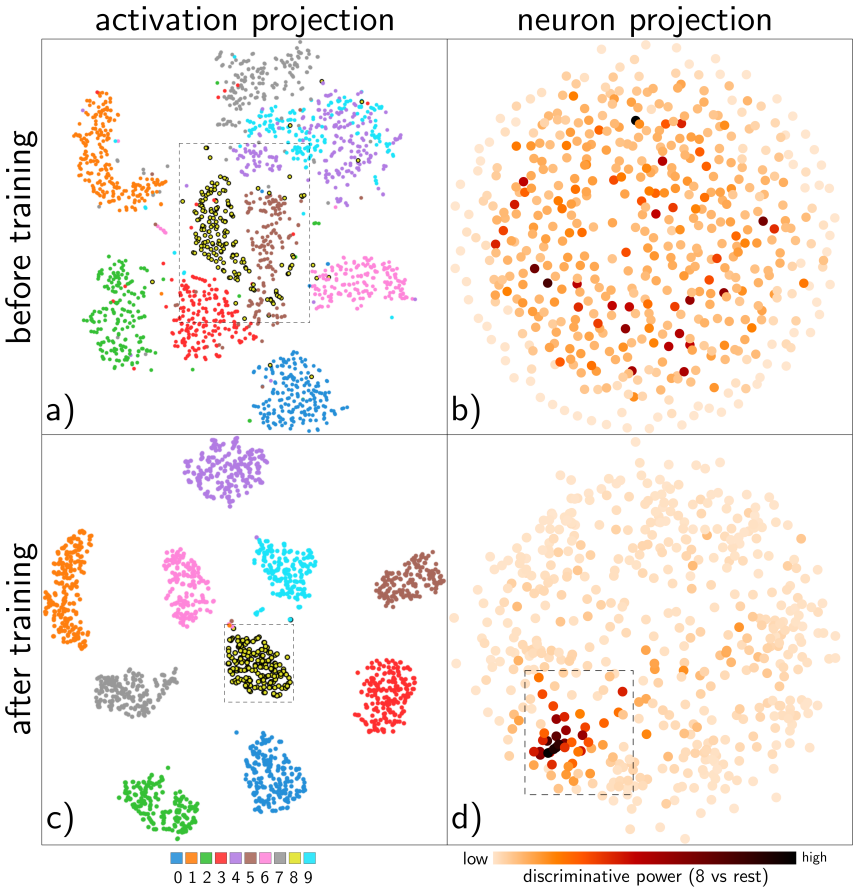


Figure 5.12: Activation and neuron projections of last CNN hidden layer activations before and after training, *MNIST* test subset. Neuron projection colors show the neurons' power to discriminate class 8 *vs* rest.

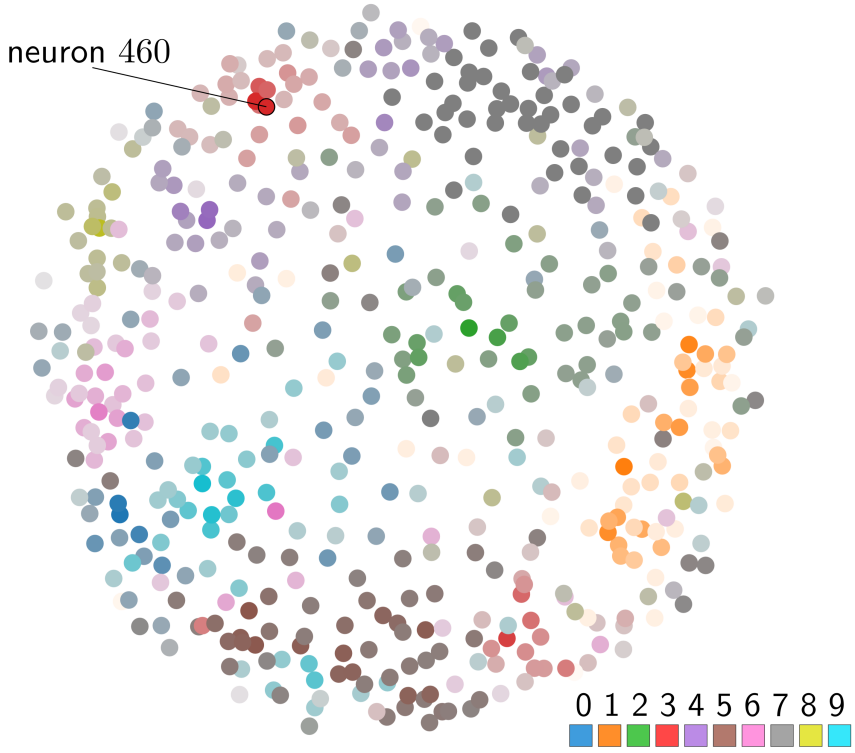


Figure 5.13: Discriminative neuron map of last CNN hidden layer activations after training, *SVHN* test subset.

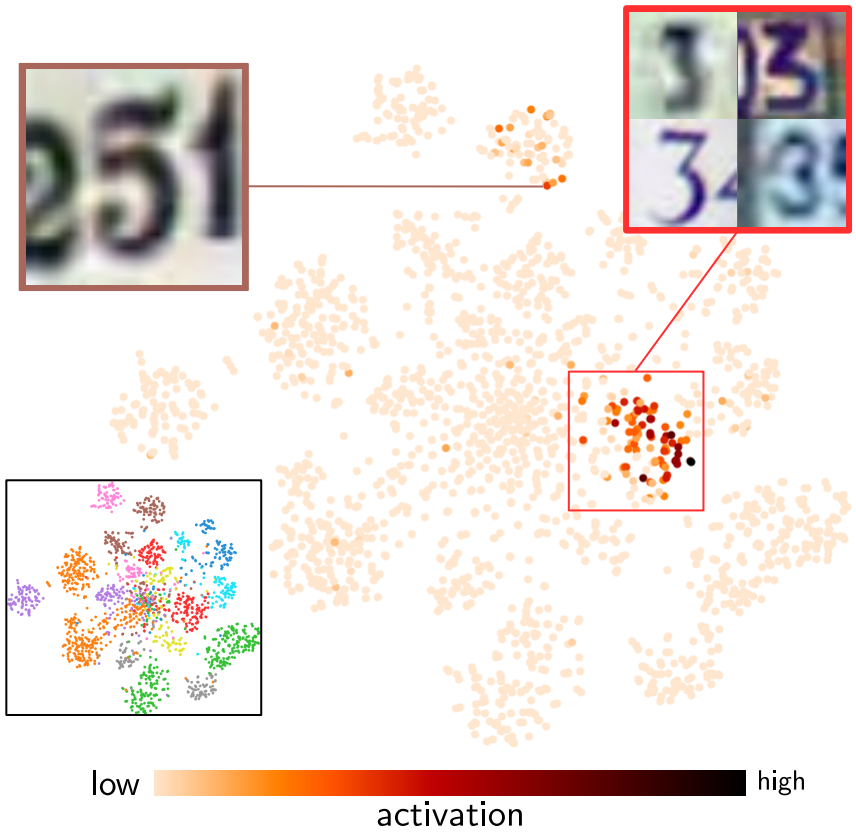


Figure 5.14: Activation projection of the last CNN hidden layer after training, *SVHN* test subset. Color shows the activation of neuron 460, highly associated to class 3 (see also Fig. 5.13).

Chapter 6

Visualizing time-dependent data using projections

In the previous chapter, we attempted to visualize *evolving* (in a broad sense, time-dependent) high-dimensional data using a standard dimensionality reduction technique. In this chapter, we will discuss the drawbacks of this approach in more detail, and propose an alternative.

Time-oriented data visualization is a widely researched subject. According to Aigner *et al.* [1], current techniques can be categorized as abstract or spatial, univariate or multivariate, linear or cyclic, instantaneous or interval-based, static or dynamic, and two or three-dimensional. Our work is concerned with abstract, multivariate, and instantaneous time-oriented visualization.

We define a time-dependent dataset as a sequence of datasets captured at particular time steps. In such a sequence, each dataset is a sequence of observations, and each observation has a corresponding observation across time steps. In simple terms, each observation *evolves* with time (or any other discrete parameter).

Consider the task of visualizing a time-dependent dataset. If a dimensionality reduction (DR) technique is applied independently for each time step, the resulting sequence of projections may present variability that does not reflect significant changes in the *structure* of the data. We refer to this issue as *temporal incoherence*, which significantly impairs the visualization of temporal trends. In this chapter, we will show that this issue affects t-SNE [148], a technique whose importance was already established in the previous chapters. Furthermore, temporal incoherence will affect any DR technique that is sensitive to relatively small changes in their inputs [49].

In this context, we also propose dynamic t-SNE: an adaptation of t-SNE that allows a controllable trade-off between temporal coherence and spatial coherence (defined as preservation of structure at a particular time step). Previous work on this trade-off has been restricted to the context of dynamic graph drawing [156, 91], even though there are many examples of time-dependent high-dimensional data visualizations based on DR [74, 13, 3]. As will become clear, our approach can be easily extended to other optimization-

This chapter is based on the following publication:
Paulo E. Rauber, Alexandre X. Falcão, and Alexandru C. Telea. Visualizing time-dependent data using dynamic t-SNE. In *EuroVis Short Papers*, 2016.

based DR techniques.

This chapter is organized as follows. Section 6.1 briefly reviews our notation and t-SNE. Section 6.2 explains the necessity for a controllable bias towards temporal coherence, and presents our proposed solution. Section 6.3 presents a preliminary evaluation of this proposal. Finally, Section 6.4 summarizes our contributions and suggests future work.

6.1 T-SNE

A dataset $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ is a sequence of observations, which are D-dimensional real vectors. The goal of t-SNE is to compute a sequence of points (projection) $\mathcal{P} = \mathbf{p}_1, \ldots, \mathbf{p}_N$ where the *neighborhoods* from $\mathcal{D}$ are preserved, considering that each $\mathbf{p}_i \in \mathbb{R}^d$ corresponds to $\mathbf{x}_i \in \mathbb{R}^D$. Typically, $d = 2$ and $D \gg d$.

T-SNE aims at minimizing a particular cost C with respect to $\mathcal{P}$. For our purposes, it suffices to note that C heavily penalizes placing *neighbors* in $\mathcal{D}$ far apart in $\mathcal{P}$. We refer to Sec. 2.6.4 for more details.

The cost C is usually minimized with respect to $\mathcal{P}$ by (momentum-based) gradient descent: from an arbitrary initial $\mathcal{P}$, for a number of iterations, each $\mathbf{p}_i \in \mathcal{P}$ is moved in the direction $-\nabla_{\mathbf{p}_i} C$.

As we explained in Sec. 2.6.4, the gradient $\nabla_{\mathbf{p}_i} C$ of C with respect to a point $\mathbf{p}_i \in \mathcal{P}$ can be interpreted as a linear combination of vectors pointing in the direction $\mathbf{p}_i - \mathbf{p}_j$, for every j. Each vector $\mathbf{p}_i - \mathbf{p}_j$ is also weighted by whether $\mathbf{p}_j$ should be moved closer to $\mathbf{p}_i$ to preserve neighborhoods from $\mathcal{D}$, and by whether $\mathbf{p}_j$ is currently close to $\mathbf{p}_i$.

6.2 Dynamic t-SNE

Consider the task of creating a sequence of projections $\mathcal{P}[1], \ldots, \mathcal{P}[T]$ for a (sequence of datasets) time-dependent dataset $\mathcal{D}[1], \ldots, \mathcal{D}[T]$, where each $\mathbf{x}_i[t] \in \mathcal{D}[t]$ corresponds to $\mathbf{x}_i[t+1] \in \mathcal{D}[t+1]$. Although we will say that the sequence of datasets represents a *time*-dependent process, this task is meaningful whenever there is correspondence between observations at different *steps*.

We will let $C[t]$ denote the usual t-SNE cost for dataset $\mathcal{D}[t]$ and projection $\mathcal{P}[t]$, as defined in Sec. 2.6.4. It is possible to apply t-SNE individually for each dataset in a sequence using at least four different strategies:

1. Initializing $\mathcal{P}[t]$ independently and randomly, for all t.
2. Initializing $\mathcal{P}[t]$ with the same random sequence, for all t.
3. Initializing $\mathcal{P}[1]$ randomly, and $\mathcal{P}[t+1]$ with the $\mathcal{P}[t]$ that results from minimizing $C[t]$, for all $t > 1$, or reversely.
4. Combining datasets from all time steps into a single dataset $\mathcal{D}$, and computing a single projection $\mathcal{P}$.

However, each of these strategies has significant drawbacks.

Strategies 1 and 2 often result in a sequence of projections with major changes in positioning of corresponding points in adjacent time steps (temporal incoherence). This issue cannot be corrected by rigid transformations (*e.g.*, rotations, translations), is misleading, and makes tracking the evolution of the data more challenging (see Sec. 6.3.1).

Strategy 3 is viable in some cases. However, it lacks a mechanism to enforce temporal coherence after initialization. At the other extreme, the initial bias may be difficult for gradient descent to overcome, because of the diminished effect on $\nabla_{\mathbf{p}_i[t]} C[t]$ of a point that is distant from $\mathbf{p}_i[t]$. These two issues also affect the (unlisted) strategy employed in the previous chapter, where we initialized $\mathcal{P}[t]$ with a projection created for $\mathcal{D}[1]$, for all t (including $t = 1$).

Furthermore, returning to strategy 3, because t-SNE usually takes many iterations to converge, the optimization of $C[t]$ starts at a likely *advantaged* state when compared to the optimization of $C[t']$, for all $t' \ll t$. In this case, the evolution due to the optimization process can be mistaken for temporal evolution.

As an extreme example, consider a particular sequence of 100 *identical* datasets, each with 2000 observations in $\mathbb{R}^{512}$. Figure 6.1 shows some projections that result from strategy 3, which are clearly misleading. Notice how there is significant *apparent* evolution between time steps 1 and 50 (10^3 and 5×10^4 gradient descent iterations, respectively). In fact, the configuration still changes between 5×10^4 and 10^5 iterations, albeit more slowly. Running t-SNE for this many iterations (for each projection) is impractical, and tweaking the parameters to achieve faster convergence is not trivial. Although it suffices to realize that there is no actual temporal evolution in this time-dependent dataset, the experimental details are described in Sections 6.3 and 6.3.2.

In summary, the major issue with strategy 3 is the lack of control over how the optimization is biased.

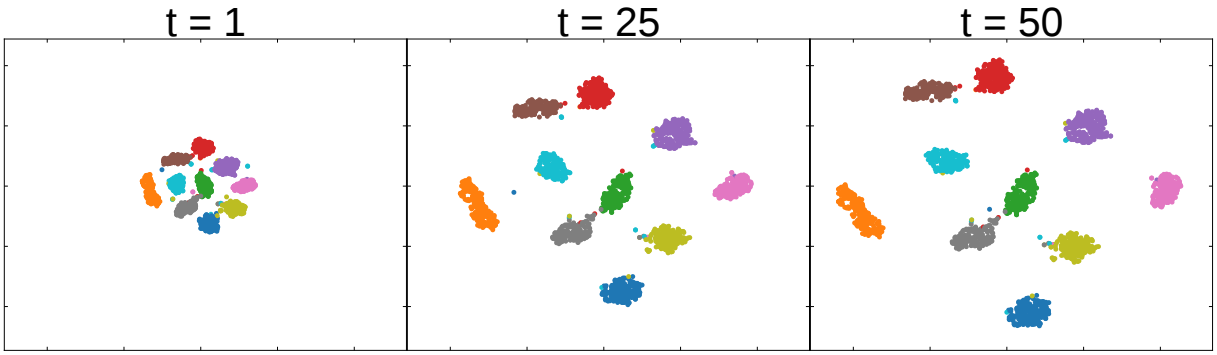


Figure 6.1: Strategy 3 results on a sequence of *identical* datasets (last CNN hidden layer fixed at epoch 1, MNIST test subset).

Strategy 4 can be dismissed in many cases. Firstly, when the distance matrix for $\mathcal{D}$ and all σ_i are given as inputs, and the target dimension d is seen as a constant, t-SNE has time complexity $O(N^2)$ for N observations. Thus, strategy 4 quickly becomes intractable. Secondly, it also introduces significant *clutter*, which cannot be eliminated

by filtering points per time step, since that introduces misleading void spaces. Finally, depending on context, combining *structures* across different steps may be inappropriate.

Dynamic t-SNE, our proposal, is an alternative that overcomes the drawbacks of the previous strategies. The dynamic t-SNE cost C tries to preserve the neighborhoods from $\mathcal{D}[t]$ in $\mathcal{P}[t]$, for each t, but also penalizes each point for unnecessarily moving between time steps. This new cost introduces a hyperparameter $\lambda \geq 0$ that *controls* the bias for temporal coherence, and is given by

$$C = \sum_{t=1}^{T} C[t] + \frac{\lambda}{2N} \sum_{i=1}^{N} \sum_{t=1}^{T-1} || \ \mathbf{p}_i[t] - \mathbf{p}_i[t+1] \ ||^2. \tag{6.1}$$

Intuitively, each point is penalized in proportion to the total squared length of the line segments formed by its movement through time. This penalty is similar to the one proposed by Leydesdorff and Schank [91] for dynamic graph drawing using multidimensional scaling (MDS).

Although it would be possible to penalize each movement in $\mathbb{R}^d$ in proportion to the corresponding movement in $\mathbb{R}^D$, that would have undesirable consequences. Firstly, supposing large variance in high-dimensional movement, it could make the choice of λ considerably more difficult. Secondly, any transformation that moved observations significantly while preserving most pairwise distances would justify significant changes in the projection. This is undesirable because the t-SNE cost depends solely on distances, which makes a projection convey only relative positioning. Despite these issues, we believe that such alternatives should be investigated in future work.

It is easy to show that the gradient $\nabla_{\mathbf{p}_i[t]} C$ of C with respect to a point $\mathbf{p}_i[t] \in \mathcal{P}[t]$ is given by

$$\nabla_{\mathbf{p}_i[t]} C = \nabla_{\mathbf{p}_i[t]} C[t] + \frac{\lambda}{N} \mathbf{v}_i[t], \tag{6.2}$$

where $\nabla_{\mathbf{p}_i[t]} C[t]$ is the usual t-SNE cost gradient (with respect to $\mathbf{p}_i[t]$) when the dataset $\mathcal{D}[t]$ is considered separately, and $\mathbf{v}_i[t]$ is given by

$$\mathbf{v}_i[t] = \begin{cases} 2\mathbf{p}_i[t] - (\mathbf{p}_i[t-1] + \mathbf{p}_i[t+1]) & \text{if } 1 < t < T, \\ \mathbf{p}_i[t] - \mathbf{p}_i[t+1] & \text{if } t = 1, \\ \mathbf{p}_i[t] - \mathbf{p}_i[t-1] & \text{if } t = T. \end{cases} \tag{6.3}$$

Just as $\nabla_{\mathbf{p}_i[t]} C[t]$, each vector $\mathbf{v}_i[t]$ also has a geometrical interpretation. For $1 < t < T$, the vector $\mathbf{v}_i[t]$ has *opposite* direction to any vector that points from $\mathbf{p}_i[t]$ to the midpoint between $\mathbf{p}_i[t-1]$ and $\mathbf{p}_i[t+1]$. Thus, in gradient *descent*, the parameter λ controls the trade-off between moving each $\mathbf{p}_i[t]$ in a direction that tries to preserve neighborhoods from $\mathcal{D}$, and moving each $\mathbf{p}_i[t]$ in a direction that minimizes the total squared length of line segments in the polyline $(\mathbf{p}_i[t-1], \mathbf{p}_i[t], \mathbf{p}_i[t+1])$.

6.3 Evaluation

We implemented t-SNE and dynamic t-SNE in Python, using Theano [9], Numpy [149], and scikit-learn [118] . Theano allows writing mathematical expressions that can be automatically translated into optimized (CPU or GPU) code and evaluated. Our implementation uses automatic differentiation, which can be highly valuable for adapting t-SNE to a particular application. For instance, altering the symbolic expression that defines the cost does not require manually finding (possibly involved) partial derivatives analytically, nor changing the optimization process. Dynamic t-SNE requires roughly the same computational time as executing t-SNE independently for each time step (Strategies 1-3). Using an *Intel i7-2600* at 3.4 GHz with a GeForce GTX 590, both (GPU) implementations require approx. 6 minutes *per time step* for the time-dependent dataset in Sec. 6.3.1.

The remainder of this section presents our preliminary experimental evaluation of dynamic t-SNE. The implementation details and hyperparameter choices are very similar to those of publicly available implementations [147]. We use momentum-based gradient descent for minimizing C, with a learning rate $\eta = 2400$ and momentum coefficient $\mu = 0.5$, which change to $\eta = 250$ and $\mu = 0.8$ at iteration 250. The optimization is run for 1000 iterations, with a perplexity $\kappa = 70$. We sample the initial coordinates of each point from a Gaussian distribution with zero mean and standard deviation 10^{-4}. The binary search for each σ_i lasts 50 iterations. For dynamic t-SNE, every projection $\mathcal{P}[t]$ is initialized equally.

6.3.1 Multivariate Gaussians

We created the multivariate Gaussians dataset specifically as a controlled experiment for dynamic t-SNE. Firstly, we sample 200 observations from each of 10 distinct (isotropic) 100-dimensional multivariate Gaussian distributions with variance 0.1. We combine the resulting 2000 observations into a single dataset $\mathcal{D}[1]$. Each multivariate Gaussian has a distinct mean, which is chosen uniformly between the standard basis vectors for $\mathbb{R}^{100}$. Given $\mathcal{D}[t]$, the dataset $\mathcal{D}[t+1]$ is created as follows. Each observation $\mathbf{x}[t+1] \in \mathcal{D}[t+1]$ corresponds to an observation $\mathbf{x}[t] \in \mathcal{D}[t]$ moved 10% of the remaining distance closer to the mean of its corresponding multivariate Gaussian. In simple terms, each of the 10 clusters becomes more *compact* as t increases. We consider $T = 10$ datasets.

The sequence of images in Fig. 6.2a shows dynamic t-SNE results for $\lambda = 0$, which corresponds to strategy 2 (as defined in Sec. 6.2). Each point $\mathbf{p}_i[t]$ is colored, for illustration purposes, according to the distribution from which $\mathbf{x}_i[1]$ was sampled. Notice the large variability in *visual cluster* positioning between time steps, even after the clusters become well-defined. Because the process that originates the data simply makes the clusters gradually more compact, this variability is misleading. We preserve the scatterplot scale between time steps, which is also a significant source of variability.

In comparison, consider the results shown in Fig. 6.2b, for $\lambda = 0.1$. Notice how each cluster stays at a similar relative position during all steps, and only becomes more

Available in `https://github.com/paulorauber/thesne`.

compact in later steps. When the projections are inspected step by step, it becomes easier to notice the movement of projection *outliers*, which is obscured when $\lambda = 0$.

Because each point is penalized for moving between projections, clear *visual separation* between clusters in later projections is also able to induce better separation in earlier projections. In simple terms, given a similar spatial coherence in two alternative projections for time step t, the projection that is more temporally coherent with the projection for time $t + 1$ is preferred by the cost function. There is a trade-off: a large λ will induce unwanted bias, whereas a small λ will cause misleading temporal incoherence. The major benefit of dynamic t-SNE is precisely the control over this trade-off. Although the choice of λ depends on context, we recommend first comparing $\lambda = 0$ with the results of an arbitrary low value.

6.3.2 Hidden layer activations

The time-dependent dataset $\mathcal{D}[0], \ldots, \mathcal{D}[T]$ considered in this section is composed of neural network activation sets, which we introduced in the previous chapter. Recall that an activation vector $\mathbf{x}[t] \in \mathcal{D}[t]$ is a D-dimensional real vector that represents the outputs of D neurons in a particular layer of an artificial neural network given a particular input. Such activation vector can be seen as an alternative representation of the input, learned by the network through an optimization process. As we have shown in the previous chapter, visualizing activation vectors allows valuable insight into how a network learns and operates, which is considered highly valuable by practitioners.

In this particular case, each network input belongs to a subset of 2000 test images from the SVHN dataset [110], a traditional image classification benchmark, and is assigned to one of ten classes (according to the digit seen in the image), which we use to color the projections (see Secs. 5.3 and 5.4.2 for more details).

For each t, an activation $\mathbf{x}[t] \in \mathcal{D}[t]$ is a 512-dimensional real vector, and corresponds to the last hidden layer activation of a convolutional neural network (CNN) after t epochs of training (given a particular input image). The time-dependent dataset represents the evolution of the learned representations through 100 epochs. Earlier in the text, the projections shown in Fig. 6.1 correspond to a similar dataset based on 2000 MNIST [86] test images, and 100 copies of the same dataset after one training epoch.

Figures 6.3a and 6.3b compare the results of dynamic t-SNE for $\lambda = 0$ and $\lambda = 0.1$, respectively. Notice that the projections for step $t = 0$, which correspond to network activations before training, are noticeably different from those that follow. Clearly, the early epochs of training have a significant effect on the learned representations, which coincides with most of the increase in validation accuracy (not shown). Although both sequences show significant variation between steps $t = 25$ and $t = 100$, the remarkable distinction is that the projections change *smoothly* when $\lambda = 0.1$. For an example, compare the transition between steps 24 and 25 in Figs. 6.3a and 6.3b. This phenomenon can be seen consistently through the whole sequence. The visual separation between clusters does not seem to improve considerably after the early epochs, although it is hard to state whether there is significant variability in the structure of the data. Because $\lambda = 0.1$ does not seem to introduce a misleading bias in comparison to $\lambda = 0$, more

evidence could be obtained by increasing λ even further.

6.4 Conclusion

In this chapter, we have shown how dynamic t-SNE can be applied to create sequences of projections with increased temporal coherence, which facilitates tracking the evolution of high-dimensional time-dependent data. The main advantage of dynamic t-SNE over t-SNE is the control over the trade-off between temporal coherence (between successive projections) and spatial coherence (with respect to high-dimensional neighborhoods). This control depends on a single hyperparameter λ, which has a simple interpretation, and does not introduce a significant computational overhead. This approach can be easily adapted for other optimization-based DR techniques. Our preliminary experiments show promising results in eliminating unnecessary variability between projections.

Although we implemented dynamic t-SNE as an adaptation of *traditional* t-SNE, the Barnes-Hut approximation is significantly more computationally efficient [146]. Future works that employ dynamic t-SNE for large datasets should consider a similar optimization. The current implementation has the advantage of being highly flexible with respect to cost functions.

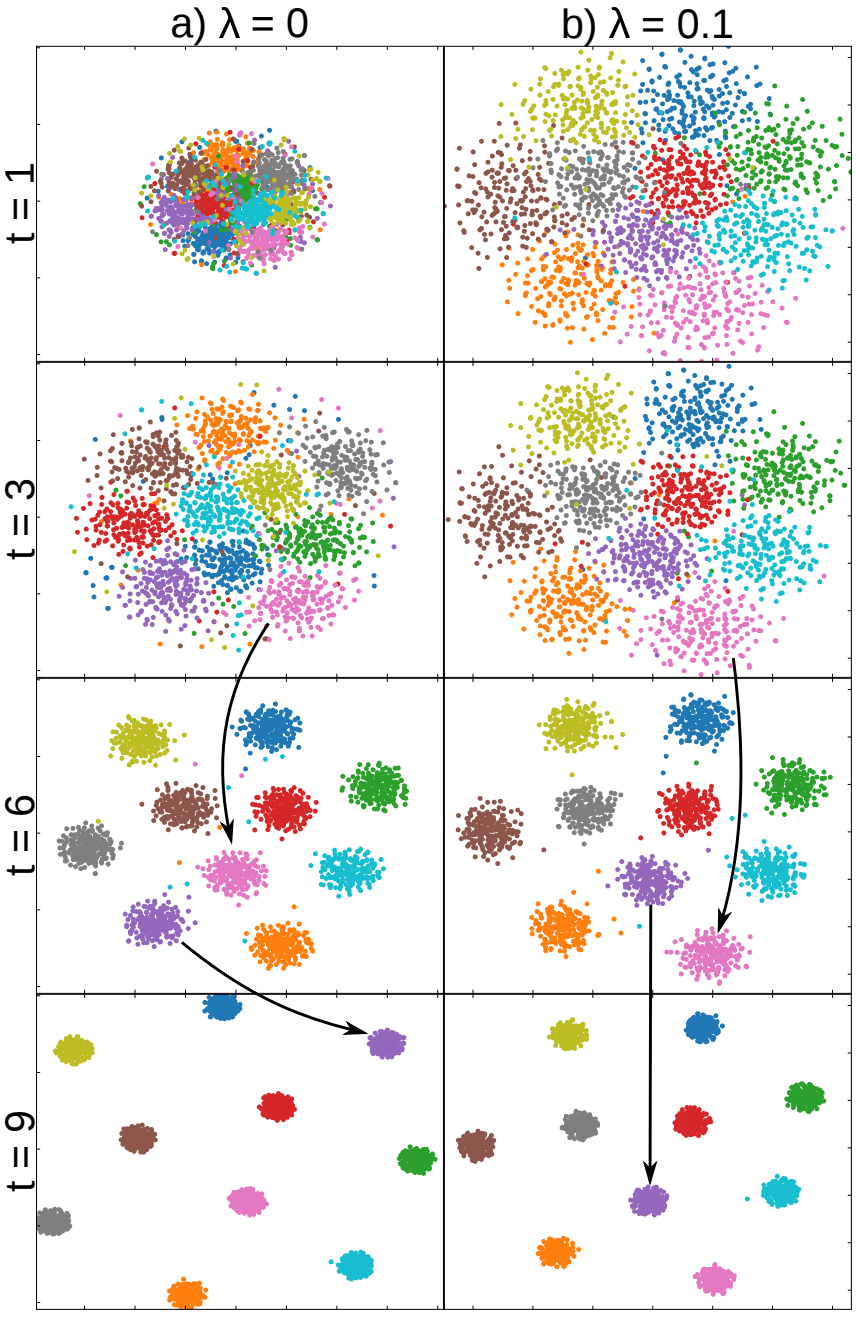


Figure 6.2: Dynamic t-SNE results on Multivariate Gaussians.

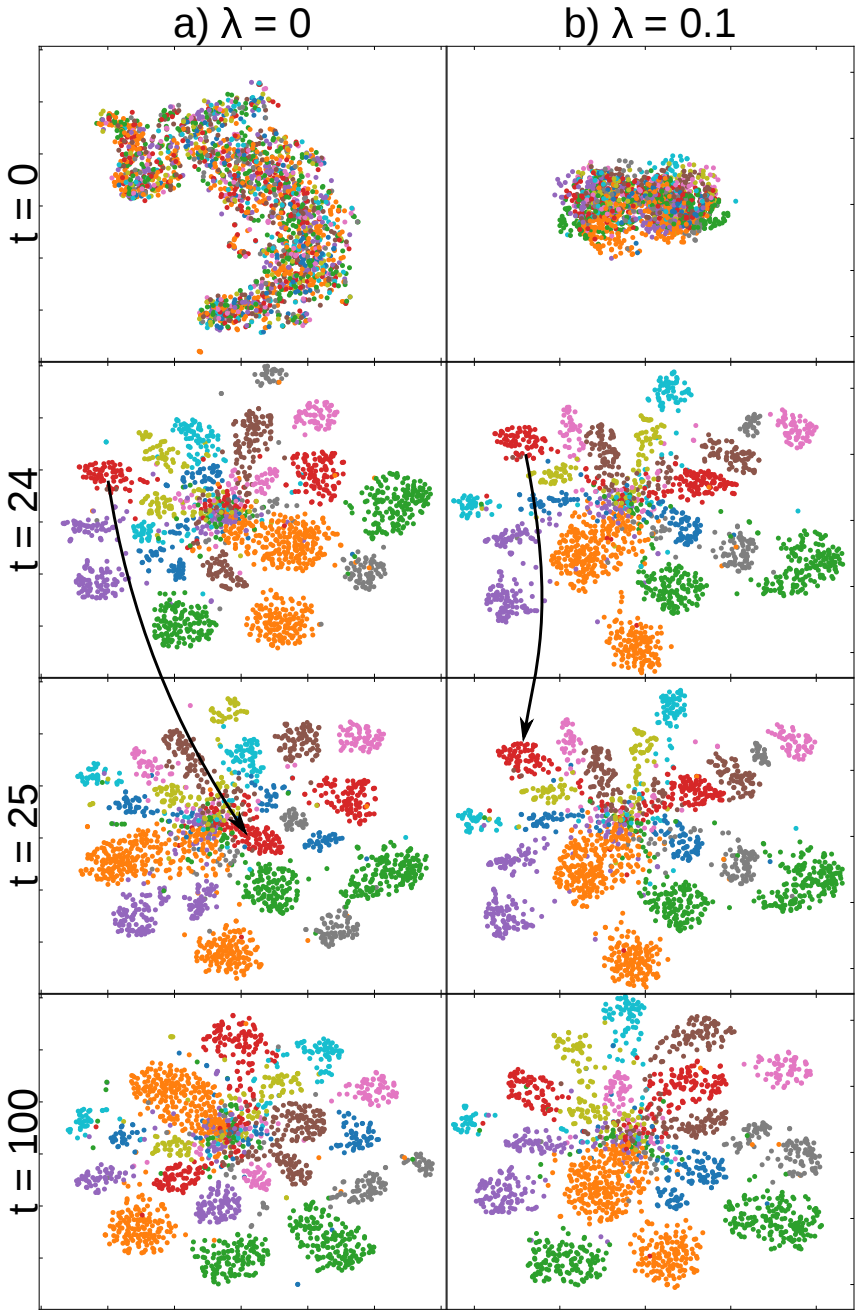


Figure 6.3: Dynamic t-SNE results on SVHN CNN.

Chapter 7

Conclusion

This thesis has explored several ways in which user interaction may assist important tasks in image analysis: image segmentation, feature selection, and image classification.

Our contributions include a new interactive segmentation technique, a new interactive visualization approach for feature selection and inspection of artificial neural networks, and a new dimensionality reduction technique for time-dependent data. These contributions are connected by the application of visual analytics, and by the prevalence of high-dimensional datasets. In this last chapter, we summarize these and other contributions (Secs. 7.1-7.4).

Regarding our research question (Sec. 1.5), we successfully showed how visual analytics can provide actionable insights about the design and operation of image analysis methods. However, as we discuss in the next sections, there are plenty of opportunities for future work.

7.1 Image segmentation

Image segmentation is a crucial task in many practical applications, and interactive methods remain indispensable when the objects of interest resist a rigorous definition.

Chapter 3 presents a new interactive segmentation technique based on superpixels that employs the IFT algorithm. This technique is motivated by the advantages that this algorithm has over other algorithms that find optimum cuts in graphs, such as linear (or linearithmic) time multi-object segmentation [26]. In comparison to its pixel-based counterpart, our new technique is significantly more (computationally) efficient for interactive segmentation. It also has the potential for exploring feature descriptors based on superpixels. Finding appropriate superpixel descriptors remains an open problem, which partially motivates our work in interactive feature selection.

Chapter 3 also introduces novel robot users, which serve as a testbed for new interactive segmentation techniques, and attempt to avoid the costs and biases involved in evaluation by real users. The relationship between these robots and real users could be studied by future work.

We omitted the description of some implemented ideas that were not sufficiently evaluated. They include supervoxel-based segmentation of volumetric images, multiscale over-

segmentation, and superpixel-based differential IFT [41], all of which could be further developed.

7.2 Image classification and feature selection

In pattern classification, representing objects of interest by observations (real vectors) is generally a challenging task. In particular, selecting features that enable good generalization is arguably harder than choosing a learning algorithm that performs reasonably well. Feature selection also affects tasks that are typically considered outside the scope of machine learning, such as image segmentation.

At the same time, dimensionality reduction is considered one of the most scalable alternatives for high-dimensional data visualization. This characteristic led us to consider using projections to assist in the difficult task of (image) classification system development.

Chapter 4 shows that projections may be useful for predicting classification system behavior. The qualitative visual feedback provided by projections allows inspecting the presence of outliers, separation between classes, and distribution of observations in the feature space. Such feedback is generally very difficult to obtain by other (non-visual) means. Although there is no guarantee that a projection will provide insightful feedback about a particular dataset, our approach requires only a small upfront user effort investment.

Chapter 4 also shows that such visual feedback may serve as a basis for a novel interactive system that assists classification system development. Our proposed system combines projections of observations and features with traditional feature selection techniques. In several use cases, this system allows eliminating a large number of candidate features from consideration, which is highly valuable for deciding where to focus effort in feature design.

One possibility for future work is employing projections as a basis to assist active learning, a process where a learning algorithm iteratively suggests which observations within a partially labeled dataset should be labeled to enable effective generalization [65, 68]. This process fits perfectly into the typical visual analytics workflow.

7.3 Image classification by artificial neural networks

Although artificial neural networks recently became able to achieve excellent raw image classification results [130], bypassing feature engineering and selection, building and training these networks generally requires significant amounts of time, expertise, and labeled data.

Chapter 5 shows how the visualization approach proposed in Chapter 4 can be adapted to visualize the relationships between learned representations of observations and between neurons in artificial neural networks. From another perspective, while Chapter 4 is concerned with inputs, Chapter 5 is concerned with intermediary computational results.

Chapter 5 shows how our approach supports the identification of confusion zones, outliers, and clusters in the learned representations of observations. It also shows how the

evolution of learned representations may be tracked using a compact trail-based visualization.

Additionally, Chapter 5 shows how our approach enables the inspection of relationships between neurons and classes (specialization), and similarity between neurons. Concrete examples show that these and the previous visualizations may prompt improvements along the classification pipeline, besides providing insight into how a network operates.

A possible direction for future work is visualizing representations learned by recurrent networks, which currently achieve state-of-the-art results in many sequence-related tasks [58].

7.4 Time-dependent data visualization

Our approach towards visualizing representations of observations learned by artificial neural networks naturally leads to our final contribution.

When a traditional dimensionality reduction technique is applied to visualize a sequence of datasets that represents a time-dependent process, the resulting sequence of projections may present variability that does not reflect significant changes in the structure of the data. This temporal incoherence affects the visualization of temporal trends, leads to incorrect insights, and will affect any traditional dimensionality reduction technique that is sensitive to relatively small changes to its inputs [49].

Chapter 6 presents dynamic t-SNE, our proposed solution to temporal incoherence in t-SNE, a state-of-the-art dimensionality reduction technique that was widely employed in Chapters 4 and 5. Our approach can also be easily adapted for other optimization-based techniques.

Dynamic t-SNE enables a trade-off between temporal coherence (between successive projections) and spatial coherence (with respect to high-dimensional neighborhoods). This trade-off depends on a coherence hyperparameter, which has a simple interpretation, and does not introduce a significant computational overhead.

Possibilities for future work include studying alternative incoherence penalty functions, and devising a principled way to choose the coherence hyperparameter.

Bibliography

[1] Wolfgang Aigner, Silvia Miksch, Heidrun Schumann, and Christian Tominski. *Visualization of time-oriented data*. Springer Science & Business Media, 2011.

[2] Davide Albanese, Roberto Visintainer, Stefano Merler, Samantha Riccadonna, Giuseppe Jurman, and Cesare Furlanello. mlpy: Machine learning python. *arXiv preprint arXiv:1202.6548*, 2012.

[3] Aretha B Alencar, Katy Börner, Fernando V Paulovich, and Maria Cristina F de Oliveira. Time-aware visualization of document collections. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 997–1004. ACM, 2012.

[4] S. Alpert, M. Galun, A. Brandt, and R. Basri. Image segmentation by probabilistic bottom-up aggregation and cue integration. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 34, pages 315–327, 2012.

[5] Giuseppe Argenziano, H Peter Soyer, Vincenzo De Giorgio, Domenico Piccolo, Paolo Carli, Mario Delfino, Angela Ferrari, Rainer Hofmann-Wellenhof, Daniela Massi, Giampero Mazzocchetti, et al. *Interactive atlas of dermoscopy*. EDRA Medical Publishing & New Media, 2000.

[6] M. Aubry and B. Russell. Understanding deep features with computer-generated imagery. In *ICCV*, 2015.

[7] Michaël Aupetit. Visualizing distortions and recovering topology in continuous projection techniques. *Neurocomputing*, 70(7):1304–1330, 2007.

[8] Sheldon Jay Axler. *Linear algebra done right*, volume 2. Springer, 1997.

[9] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

[10] R. Becker, W. Cleveland, and M. Shyu. The visual design and control of trellis display. *J. Comp. Graph. Stat*, 5:123–155, 1996.

[11] Yoshua Bengio. Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 2(1):1–127, 2009.

[12] Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[13] Jürgen Bernard, Nils Wilhelm, Maximilian Scherer, Thorsten May, and Tobias Schreck. Timeseriespaths : Projection-based explorative analysis of multivariate time series data. In *Journal of WSCG*, pages 97–106, 2012.

[14] Dimitri P Bertsekas. *Nonlinear programming*. Athena scientific, 1999.

[15] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[16] Andrew Blake, Carsten Rother, M. Brown, Patrick Pérez, and Philip H. S. Torr. Interactive image segmentation using an adaptive GMMRF model. In *European Conference on Computer Vision (ECCV)*, pages 428–441, 2004.

[17] Ingwer Borg and Patrick JF Groenen. *Modern multidimensional scaling: Theory and applications*. Springer Science & Business Media, 2005.

[18] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.

[19] Y.Y. Boykov and M.-P. Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in N-D images. In *Computer Vision, Eighth IEEE International Conference on*, volume 1, pages 105–112, 2001.

[20] Bruno Brandoli, Danilo Eler, Fernando Paulovich, Rosane Minghim, and Joao Batista. Visual data exploration to feature space definition. In *Graphics, Patterns and Images (SIBGRAPI), 2010 23rd SIBGRAPI Conference on*, pages 32–39. IEEE, 2010.

[21] M. Brehmer, M. Sedlmair, S. Ingram, and T. Munzner. Visualizing dimensionally-reduced data: Interviews with analysts and a characterization of task sequences. In *Proc. IEEE BELIV*, pages 68–76, 2014.

[22] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. The Wadsworth and Brooks-Cole statistics-probability series. Taylor & Francis, 1984.

[23] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[24] J. Caldas, N. Gehlenborg, A. Faisal, A. Brazma, and S. Kaski. Probabilistic retrieval and visualization of biologically relevant microarray experiments. *Bioinformatics*, 25(12):145–153, 2009.

[25] L. Chen and A. Buja. Stress functions for nonlinear dimension reduction, proximity analysis, and graph drawing. *JMLR*, 14:1145–1173, 2013.

[26] KrzysztofChris Ciesielski, JayaramK. Udupa, A.X. Falcão, and P.A.V. Miranda. Fuzzy connectedness image segmentation in graph cut formulation: A linear-time algorithm and a comparative analysis. *Journal of Mathematical Imaging and Vision*, 44:375–398, 2012.

[27] Danilo B Coimbra, Rafael M Martins, Tácito TAT Neves, Alexandru C Telea, and Fernando V Paulovich. Explaining three-dimensional dimensionality reduction plots. *Information Visualization*, 15(2):154–172, 2016.

[28] T. F. Cootes, C. J. Taylor, D. H. Cooper, and J. Graham. Active shape models – their training and application. *Comput. Vis. Image Underst.*, 61(1):38–59, 1995.

[29] Gregory W Corder and Dale I Foreman. *Nonparametric statistics: a step-by-step approach*. John Wiley & Sons, 2014.

[30] John P Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research*, 2015.

[31] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2), 2008.

[32] P. A. V. de Miranda, A. X. Falcão, and J. K. Udupa. Synergistic arc-weight estimation for interactive image segmentation using graphs. *Comput. Vis. Image Underst.*, 114(1):85–99, 2010.

[33] Thomas Deselaers, Daniel Keysers, and Hermann Ney. Features for image retrieval: an experimental comparison. *Information Retrieval*, 11(2):77–107, 2008.

[34] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[35] P. Domingos. A few useful things to know about machine learning. *Comm ACM*, 55(10):78–87, 2012.

[36] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *arXiv preprint arXiv:1310.1531*, 2013.

[37] N. Duta and M. Sonka. Segmentation and interpretation of MR brain images. an improved active shape model. *Medical Imaging, IEEE Transactions on*, 17(6):1049–1062, 1998.

[38] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *The Journal of Machine Learning Research*, 11:625–660, 2010.

[39] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *Dept. IRO, Université de Montréal, Tech. Rep*, 4323, 2009.

[40] S. Fadel, F. Fatore, F. Duarte, and F. Paulovich. LoCH: A neighborhood-based multidimensional projection technique for high-dimensional sparse spaces. *Neurocomputing*, 150:546–556, 2014.

[41] A.X. Falcão and F.P.G. Bergo. Interactive volume segmentation with differential image foresting transforms. *Medical Imaging, IEEE Transactions on*, 23(9):1100–1108, sept. 2004.

[42] A.X. Falcão, J. Stolfi, and R. de Alencar Lotufo. The image foresting transform: theory, algorithms, and applications. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(1):19–29, 2004.

[43] A.X. Falcão, J.K. Udupa, and F.K. Miyazawa. An ultra-fast user-steered image segmentation paradigm: live wire on the fly. *Medical Imaging, IEEE Transactions on*, 19(1):55–62, jan. 2000.

[44] A. X. Falcão, J. K. Udupa, S. Samarasekera, S. Sharma, B. E. Hirsch, and R. A. Lotufo. User-steered image segmentation paradigms: Live-wire and live-lane. *Graphical Models and Image Processing*, 60(4):233–260, 1998.

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

[46] Sander Feringa. Comparison of features used in automatic skin lesion classification. Master's thesis, Rijksuniversiteit Groningen, the Netherlands, 2015.

[47] K. R. Gabriel. The biplot-graphical display of matrices with applications to principal components analysis. *Biometrika*, 58:453–467, 1971.

[48] Klaus A. Ganser, Hartmut Dickhaus, Roland Metzner, and Christian R. Wirtz. A deformable digital brain atlas system according to Talairach and Tournoux. *Medical Image Analysis*, 8(1):3–22, 2004.

[49] Francisco Javier García Fernández, Michel Verleysen, John A Lee, and Ignacio Díaz Blanco. Stability comparison of dimensionality reduction techniques attending to data and parameter variations. In *Eurographics Conference on Visualization*. The Eurographics Association, 2013.

[50] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. *Bayesian data analysis*, volume 2. Taylor & Francis, 2014.

[51] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[52] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[53] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[54] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.

[55] John C. Gower and Garmt B. Dijksterhuis. *Procrustes Problems*. Oxford Statistical Science Series 30, jan 2004.

[56] V. Grau, A.U.J. Mewes, M. Alcaniz, R. Kikinis, and S.K. Warfield. Improved watershed transform for medical image segmentation using prior information. *Medical Imaging, IEEE Transactions on*, 23(4):447–458, april 2004.

[57] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[58] Alex Graves et al. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[59] V. Gulshan, C. Rother, Antonio Criminisi, A. Blake, and A. Zisserman. Geodesic star convexity for interactive image segmentation. In *Computer Vision and Pattern Recognition (CVPR), IEEE Conference on*, pages 3129–3136, 2010.

[60] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[61] Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the NIPS 2003 feature selection challenge. In *Advances in Neural Information Processing Systems*, pages 545–552, 2004.

[62] Isabelle Guyon, Jason Weston, Stephen Barnhill, and Vladimir Vapnik. Gene selection for cancer classification using support vector machines. *Machine Learning*, 46(1-3):389–422, 2002.

[63] Philippe Hamel and Douglas Eck. Learning features from music audio with deep belief networks. In *Proc. ISMIR*, pages 339–344, 2010.

[64] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer Series in Statistics. Springer, 2009.

[65] Florian Heimerl, Steffen Koch, Harald Bosch, and Thomas Ertl. Visual classifier training for text document retrieval. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2839–2848, 2012.

[66] J Heinrich, J Stasko, and D Weiskopf. The parallel coordinates matrix. *EuroVis–Short Papers*, pages 37–41, 2012.

[67] Julian Heinrich and Daniel Weiskopf. State of the art of parallel coordinates. In *Eurographics State of the Art Reports*, pages 95–116, 2013.

[68] Benjamin Hoferlin, Rudolf Netzel, Markus Hoferlin, Daniel Weiskopf, and Gunther Heidemann. Inter-active learning of ad-hoc classifiers for video visual analytics. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 23–32. IEEE, 2012.

[69] P. Hoffman, G. Grinstein, K. Marx, I. Grosse, and E. Stanley. DNA visual and analytic data mining. In *Proc. IEEE Visualization*, pages 437–441, 1997.

[70] John D Hunter. Matplotlib: A 2D graphics environment. *Computing in science and engineering*, 9(3):90–95, 2007.

[71] C. Hurter, O. Ersoy, S. Fabrikant, T. Klein, and A. Telea. Bundled visualization of dynamic graph and trail data. *IEEE TVCG*, 20(8):1141–1154, 2014.

[72] Laurent Hyafil and Ronald L Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[73] H. Iyatomi, H. Oka, M. Celebi, M. Hashimoto, M. Hagiwara, M. Tanaka, and K. Ogawa. An improved internet-based melanoma screening system with dermatologist-like tumor area extraction algorithm. *Comp Med Imag Gr*, 32(7):566–579, 200820.

[74] Dominik Jäckle, Fabian Fischer, Tobias Schreck, and Daniel A Keim. Temporal MDS plots for analysis of multivariate data. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):141–150, 2016.

[75] Paulo Joia, Fernando V Paulovich, Danilo Coimbra, José Alberto Cuminato, and Luis Gustavo Nonato. Local affine multidimensional projection. *IEEE TVCG*, 17(12):2563–2571, 2011.

[76] Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python, 2014. http://www.scipy.org.

[77] E. Kandogan. Star coordinates: A multi-dimensional visualization technique with uniform treatment of dimensions. In *Proc. IEEE Infovis*, pages 9–12, 2000.

[78] D. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, editors. *Mastering the information age: Solving problems with visual analytics (VisMaster)*. Eurographics Association, 2010.

[79] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 14, pages 1137–1145, 1995.

[80] Pushmeet Kohli, Hannes Nickisch, Carsten Rother, and Christoph Rhemann. User-centric learning and evaluation of interactive segmentation systems. *Int. J. Computer Vision*, 100(3):261–274, 2012.

[81] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[82] Jan Krause, Adam Perer, and Enrico Bertini. Infuse: interactive feature selection for predictive modeling of high dimensional data. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):1614–1623, 2014.

[83] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images, 2009. `www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

[84] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[85] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[86] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The MNIST database of handwritten digits, 1998. Available at http://http://yann.lecun.com/exdb/mnist.

[87] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. *arXiv preprint arXiv:1409.5185*, 2014.

[88] John Aldo Lee and Michel Verleysen. Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67:29–53, 2005.

[89] Shane Legg. *Machine Super Intelligence*. PhD thesis, University of Lugano, 2008.

[90] Dirk J Lehmann and Holger Theisel. Orthographic star coordinates. *Visualization and Computer Graphics, IEEE Transactions on*, 19(12):2615–2624, 2013.

[91] Loet Leydesdorff and Thomas Schank. Dynamic animations of journal maps: Indicators of structural changes and interdisciplinary developments. *Journal of the American Society for Information Science and Technology*, 59(11):1810–1818, 2008.

[92] Jia Li and James Z Wang. Automatic linguistic indexing of pictures by a statistical modeling approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(9):1075–1088, 2003.

[93] Huan Liu and Lei Yu. Toward integrating feature selection algorithms for classification and clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 17(4):491–502, 2005.

[94] Jiamin Liu and J.K. Udupa. Oriented active shape models. *Medical Imaging, IEEE Transactions on*, 28(4):571–584, 2009.

[95] S Liu, B Wang, P-T Bremer, and V Pascucci. Distortion-guided structure-driven interactive exploration of high-dimensional data. *CGF*, 33(3):101–110, 2014.

[96] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. In *Proc. EuroVis – STARs*, 2015.

[97] R.A. Lotufo, A.X. Falcão, and F.A. Zampirolli. IFT-Watershed from gray-scale marker. In *Computer Graphics and Image Processing, 2002. Proceedings. XV Brazilian Symposium on*, pages 146–152, 2002.

[98] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[99] Aravindh Mahendran and Andrea Vedaldi. Understanding deep image representations by inverting them. In *Proc. IEEE CVPR*, 2015.

[100] Filip Malmberg and Robin Strand. Faster fuzzy connectedness via precomputation. In *Mathematical Morphology and its Applications to Image and Signal Processing (ISMM)*, 2013, in press.

[101] Rafael Messias Martins, Danilo Barbosa Coimbra, Rosane Minghim, and AC Telea. Visual analysis of dimensionality reduction quality for parameterized projections. *Computers & Graphics*, 41:26–42, 2014.

[102] Rafael Messias Martins, Rosane Minghim, and Alexandru C. Telea. Explaining neighborhood preservation for multidimensional projections. In *Proc. Computer Graphics and Visual Computing (CGVC), 2015*, 2015.

[103] Nicolai Meinshausen and Peter Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010.

[104] P.A.V. Miranda, A.X. Falcão, and J.K. Udupa. Clouds: A model for synergistic image segmentation. In *Biomedical Imaging: From Nano to Macro, 5th IEEE International Symposium on*, pages 209–212, 2008.

[105] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[106] Abdel-rahman Mohamed, Geoffrey Hinton, and Gerald Penn. Understanding how deep belief networks perform acoustic modelling. In *Proc. IEEE ICASSP*, pages 4273–4276, 2012.

[107] T. Mühlbacher, H. Piringer, S. Gratzl, M. Sedlmair, and M. Streit. Opening the black box: Strategies for increased user involvement in existing algorithm implementations. *IEEE TVCG*, 20(12):1643–1652, 2014.

[108] Kevin P Murphy. *Machine learning: a Probabilistic Perspective*. MIT Press, 2012.

[109] L. Najman and M. Couprie. Building the component tree in quasi-linear time. *Image Processing, IEEE Transactions on*, 15(11):3531–3539, 2006.

[110] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *Proc. NIPS*, volume 2011, page 5, 2011.

[111] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 427–436. IEEE, 2015.

[112] Michael A Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[113] Genevieve B Orr and Klaus-Robert Müller. *Neural networks: tricks of the trade*. Springer, 2003.

[114] Jose Gustavo S Paiva, William Robson Schwartz, Helio Pedrini, and Rosane Minghim. An approach to supporting incremental visual data classification. *Visualization and Computer Graphics, IEEE Transactions on*, 21(1):4–17, 2015.

[115] J. P. Papa, A. X. Falcão, and C. T. N. Suzuki. Supervised pattern classification based on optimum-path forest. *International Journal of Imaging Systems and Technology*, 19(2):120–131, 2009.

[116] Fernando V Paulovich, Luis Gustavo Nonato, Rosane Minghim, and Haim Levkowitz. Least square projection: A fast high-precision multidimensional projection technique and its application to document mapping. *IEEE TVCG*, 14(3):564–575, 2008.

[117] Fernando V Paulovich, Franklina Toledo, Guilherme P Telles, Rosane Minghim, and Luis Gustavo Nonato. Semantic wordification of document collections. In *Computer Graphics Forum*, volume 31, pages 1145–1153. Wiley Online Library, 2012.

[118] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *J Mach Learn Res*, 12:2825–2830, 2011.

[119] Lutz Prechelt. Early stopping-but when? In *Neural Networks: Tricks of the trade*, pages 55–69. Springer, 1998.

[120] A. Protiere and G. Sapiro. Interactive image segmentation via adaptive weighted distances. *Image Processing, IEEE Transactions on*, 16(4):1046–1057, 2007.

[121] R. Rao and S. K. Card. The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information. In *Proc. ACM CHI*, pages 318–322, 1994.

[122] Paulo E. Rauber, Samuel G. Fadel, Alexandre X. Falcão, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of the Visual Analytics Science and Technology 2016)*, 23(01), January 2017.

[123] Paulo E. Rauber, Alexandre X. Falcão, Thiago V. Spina, and Pedro J. de Rezende. Interactive segmentation by image foresting transform on superpixel graphs. In *Proceedings of the 2013 XXVI Conference on Graphics, Patterns and Images*, SIBGRAPI '13, pages 131–138, Washington, DC, USA, 2013. IEEE Computer Society.

[124] Paulo E. Rauber, Alexandre X. Falcão, and Alexandru C Telea. Projections as visual aids for classification system design, 2016. Submitted to Information Visualization (IVI).

[125] Paulo E. Rauber, Alexandre X. Falcão, and Alexandru C. Telea. Visualizing time-dependent data using dynamic t-SNE. In *EuroVis Short Papers*, 2016.

[126] Paulo Eduardo Rauber, Renato Rodrigues Oliveira Silva, Sander Feringa, M. Emre Celebi, Alexandre X. Falcão, and Alexandru C. Telea. Interactive Image Feature Selection Aided by Dimensionality Reduction. In *EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association, 2015.

[127] Leonardo Marques Rocha, Fábio A. M. Cappabianco, and Alexandre Xavier Falcão. Data clustering as an optimum-path forest problem with applications in image analysis. *International Journal of Imaging Systems and Technology*, 19(2):50–68, 2009.

[128] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: interactive foreground extraction using iterated graph cuts. In *ACM SIGGRAPH 2004 Papers*, pages 309–314, New York, NY, USA, 2004. ACM.

[129] Priscila TM Saito, Pedro J de Rezende, Alexandre X Falcão, Celso TN Suzuki, and Jancarlo F Gomes. Improving active learning with sharp data reduction. In *Proc. of the 20th Intl. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)*, pages 27–34, 2012.

[130] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[131] Tobias Schreck, Tatiana Von Landesberger, and Sebastian Bremm. Techniques for precision-based visual analysis of projected data. *Information Visualization*, 9(3):181–193, 2010.

[132] Alexander Schulz, Andrej Gisbrecht, and Barbara Hammer. Using discriminative dimensionality reduction to visualize classifiers. *Neural Processing Letters*, pages 1–28, 2014.

[133] M. Sedlmair, T. Munzner, and M. Tory. Empirical guidance on scatterplot and dimension reduction technique choices. *IEEE TVCG*, 19(12):2634–2643, 2013.

[134] Renato R. O. Silva, Paulo E. Rauber, Rafael M. Martins, Rosane Minghim, and Alexandru C. Telea. Attribute-based visual explanation of multidimensional projections. In *EuroVis Workshop on Visual Analytics (EuroVA)*. The Eurographics Association, 2015.

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

[136] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[137] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J Mach Learning Res*, 15(1):1929–1958, 2014.

[138] Celso TN Suzuki, Jancarlo F Gomes, Alexandre X Falcão, Joao Paulo Papa, and Sumie Hoshino-Shimizu. Automatic segmentation and classification of human intestinal parasites from microscopy images. *Biomedical Engineering, IEEE Transactions on*, 60(3):803–812, 2013.

[139] Gábor J Székely, Maria L Rizzo, Nail K Bakirov, et al. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6):2769–2794, 2007.

[140] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[141] Aditya Tatu, F Maas, Ines Farber, Enrico Bertini, Tobias Schreck, Thomas Seidl, and Daniel Keim. Subspace search and visualization to make sense of alternative clusterings in high-dimensional data. In *Visual Analytics Science and Technology (VAST), 2012 IEEE Conference on*, pages 63–72. IEEE, 2012.

[142] Alexandru Telea. Combining extended table lens and treemap techniques for visualizing tabular data. In *Proceeding of the 2007 Eurographics Conference on Visualization (EuroVis 2007)*, pages 51–58, 2007.

[143] Joseph Tighe and Svetlana Lazebnik. Superparsing - scalable nonparametric image parsing with superpixels. *International Journal of Computer Vision*, 101(2):329–349, 2013.

[144] Cagatay Turkay, Peter Filzmoser, and Helwig Hauser. Brushing dimensions - a dual visual analysis model for high-dimensional data. *IEEE TVCG*, 17(12):2591–2599, 2011.

[145] Stef van den Elzen and Jarke J Van Wijk. Baobabview: Interactive construction and analysis of decision trees. In *Visual Analytics Science and Technology (VAST), 2011 IEEE Conference on*, pages 151–160. IEEE, 2011.

[146] Laurens Van Der Maaten. Accelerating t-SNE using tree-based algorithms. *The Journal of Machine Learning Research*, 15(1):3221–3245, 2014.

[147] Laurens Van Der Maaten. t-SNE - Laurens van der Maaten, 2016. Available at https://lvdmaaten.github.io/tsne.

[148] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[149] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[150] Stefan Van Der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[151] M. van der Zwan, V. Codreanu, and A. Telea. Cubu: Universal real-time bundling for large graphs. *IEEE Transactions on Visualization and Computer Graphics*, PP(99):1–1, 2016.

[152] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proc. ICML*, pages 1058–1066, 2013.

[153] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[154] David H Wolpert. The lack of a priori distinctions between learning algorithms. *Neural computation*, 8(7):1341–1390, 1996.

[155] Chenliang Xu and Jason J. Corso. Evaluation of super-voxel methods for early video processing. In *Computer Vision and Pattern Recognition (CVPR)*, pages 1202–1209, 2012.

[156] Kevin S Xu, Mark Kliger, and Alfred O Hero Iii. A regularized graph layout framework for dynamic network visualization. *Data Mining and Knowledge Discovery*, 27(1):84–116, 2013.

[157] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. In *Proc. ICML*, 2015.

[158] Xiaoru Yuan, Donghao Ren, Zuchao Wang, and Cong Guo. Dimension projection matrix/tree: Interactive subspace visual exploration and analysis of high dimensional data. *IEEE TVCG*, 19(12):2625–2633, 2013.

[159] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Proc. ECCV*, pages 818–833. Springer, 2014.