An Invitation to

Image Analysis and Pattern Recognition

Fred A. Hamprecht
University of Heidelberg
Version of September 26, 2010

Contents

So far, this manuscript comprises only the first of six scheduled parts. There are hence many unresolved forward references "(??)" to yet unwritten parts (there be dragons). I make part 1 available anyway in the hope that it may be useful.

1 Introduction

1.1 Overview

This script will take you on a guided tour to the intriguing world of "pattern recognition": the art and science of analyzing data. Our focus will be the automated analysis of objects or processes whose information content and complexity at least partly reside in their spatial layout. But before delving into technical details, this first chapter will take a step back, allowing to put things into perspective: to remember what the value of vision is (section 1.2), to argue why computer vision is an important and a difficult subject (section 1.3), and to summarize the categories of tasks that will be treated in this script (section 1.4).

1.2 Animate vision

We tend to not think about it, but: vision is almost too good to be true! It is indeed a magnificent sense which transcends the narrow confines of our own physical extent; which is, by all means and purposes, instantaneous; and which allows us, "in the blink of an eye", to make sense of complex scenes and react adequately. The reason we tend not to think about it is that we are endowed with vision almost from the moment we are born; and that it is an unconscious activity: we cannot help but see.

Actually, we compose, rather than perceive, a visual picture of our world: one token of evidence is given by geometrical optics which shows that the world is projected on our retina upside down. This is where our light receptors lie and yet we have the impression that the world is standing "on its feet". This seeming contradiction was cited to refute Johannes Kepler's proposal that the eye acts like a camera obscura. More evidence for the active construction of our percepts is given by optical illusions: these provide a vivid demonstration of the fact that our innate or acquired vision mechanisms help explain the world most of the time, but do sometimes lead us astray, see Figs. 1.1, 1.3.

Hence vision is an "active", if involuntary, achievement in the perceptual sense, but "passive" in a physical sense: we simply capitalize on the copious number of photons reflected or emitted by an object. The latter notion is relatively recent: from the times of Greek physician Empedocles until the 17th century or so, the prevailing belief was that the eye is endowed with

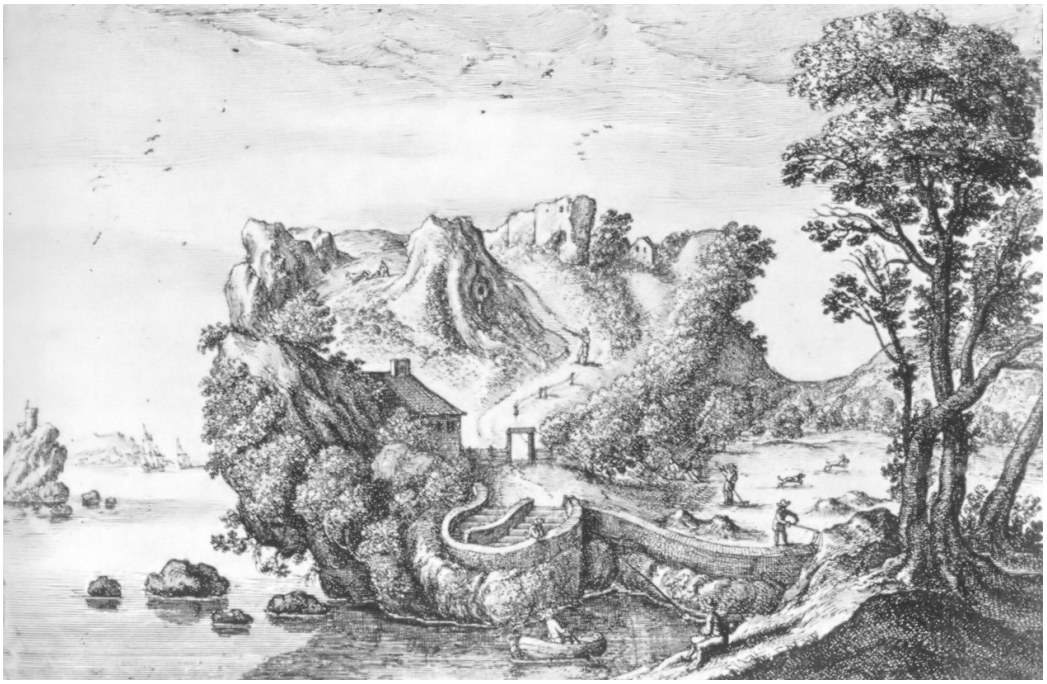


Figure 1.1: Sample of Renaissance metamorphic mannerism, by Wenceslaus Hollar (1607-1677). It shows that what we see is not what we get: sometimes more, as in this case, and sometimes less, see next figure.

Figure 1.2: Camouflage picture of at least eight commandos by contemporary artist Derek Bacon.

Figure 1.3: Example of false perspective masterpiece created by Francesco Borromini (1599-1667). A rising floor and other cues insinuate a life sized statue at the end of the gallery. In actual fact, the statue rises to a mere 60cm.

an "internal fire" and also emits some kind of radiation, rather than merely records incident light [11].

Human vision offers great spatial resolution; and while some species have evolved alternate senses (think bats), vision is unparalleled in the velocity of its stimuli – the speed of light – and in its range.

It is only too natural, then, that one would try and endow computers or machines with at least a rudimentary form of vision. However, the true complexity of this exercise that we perform continuously and ostensibly without effort becomes apparent when trying to emulate some form of visual perception in computers: it turns out to be a formidable task, as anybody who has ever tried will confirm.

1.3 Why image processing is difficult

To us, *having* a picture amounts to *understanding* it. Admittedly, interpretations often differ between individuals, and surely you have experienced examples where your own perception of an image has changed over time, as a function of additional knowledge or experiences acquired. This perceived immediacy of our own image understanding often leads us to underestimate

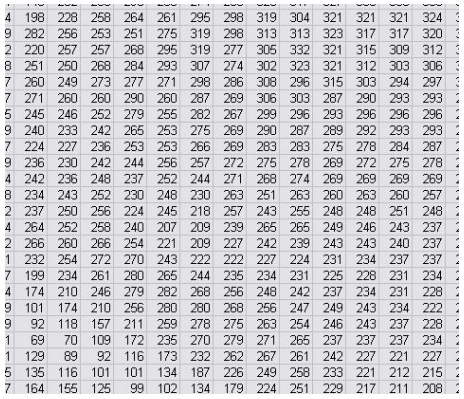


Figure 1.4: This is how a computer "perceives" an image: as a bunch of numbers.

the complexity of image analysis, let alone scene understanding.

In contrast to ourselves, the computer does not "understand" a digital picture that it stores with so much more perfection than we ever can. To the computer, any picture looks as shown in Fig. 1.4. As far as a computer is concerned, some images may allow for a higher ratio of compression than others, but other than that they are all equally meaningless.

Assume, for a moment, that our task is to write an algorithm that counts the number of cells labeled with a fluorescent marker in a large number of microscopic images. Assuming that each picture has the modest resolution of 1000×1000 pixels, the task then is, formally, to learn the mapping $f : \mathbb{R}^{1000\times1000} \to \mathbb{N}$ which should yield the correct number of cells for each image, zero if there are none. To learn a mapping f defined over a small space is often feasible; but the space of all conceivable images is very large indeed, as the following parable illustrates.

Borrowing from Jorge Luis Borges (18991986) fabulous short story "The Library of Babel", let us consider The Picture Album of Babel: the album is *total* in the sense that it contains *all* conceivable images with a resolution of, say, 1 million pixels and $256^3 = 16777216$ color or intensity values per pixel. The album contains $16777216^{1'000'000}$ which is more than $10^{7'000'000}$ images. Needless to say, parts of the album are rather interesting: a minuscule fraction of it is filled with images that show the evolution of our universe, at each instance in time, from every conceivable perspective, using every field of view from less than a nanometer to more than a megaparsec. Some of these images will show historic events, such as the moment when your great-grandmother first beheld your great-grandfather; while others will show timeless pieces of art, or detailed construction directions for future computing machinery or locomotion engines that really work. Other images yet will contain blueprints for time machines, or visual proofs attesting to their impossibility, or will show your great-grandmother riding a magic

(a) **(b)**

Figure 1.5: **(a)** An excerpt of a picture from The Album of Babel **(b)** Another picture from The Album, borrowed by contemporary artist Gerhard Richter for his stained glass design for the Cologne Cathedral.

carpet, etc.

However, to us the majority by far of these images will look like the excerpt shown in Fig. 1.5, making it a little difficult to browse through it systematically in order to find the "interesting" images, such as the one showing the blueprint of the next processor generation which could earn you a lot of money. The album is so huge that, were you to order all offprints from the album at the modest size of 9×13cm, you would need a shoe box about $10^{1'999'913}$ times the size of the visible universe to file them.

The good news is that the images of interest to us only make up a tiny part of this total album; but even so, building a general-purpose algorithm that will do well in such a high-dimensional space is hard to build.

1.4 Tasks in computer vision

So far, we have shied from defining what "computer vision" actually is. One possible taxonomy is in terms of

- image processing
- image analysis
- computer vision or image understanding / scene understanding.

All these operate on images or videos, which are usually represented in terms of arrays: a monochrome still image is represented as a two-dimensional matrix; a color or spectral image is represented as a three-dimensional array;

and a color video has two spatial dimensions, one color dimension and one temporal dimension and can hence be indexed as a four-dimensional array.

In **image processing**, input and output arrays have the same dimensions, and often the same number of values. In **image analysis**, the input is an image or video and the output is a typically much smaller set of features (such as the number of cells in an image, the position of a pedestrian, or the presence of a defect). In **computer vision**, the input is an image or video, while the output is a high-level semantic description or annotation, perhaps in terms of a complex ontology or even natural language. Even after five decades of research, computer vision is still in its infancy. Image processing, in contrast, is a mature subject which is treated in excellent textbooks such as [15]. The present script is mainly concerned with image analysis.

Typical tasks in image processing and analysis include:

- **image restoration**, to make up for deteriorations caused by deficient optics, motion blur, or suboptimal exposure, (**??**)
- **object detection, localization** and **tracking** (**??**)
- estimation of **pose, shape, geometry** with applications such as human-machine interaction, robotics and metrology (**??**)
- estimation of **motion or flow**
- **extraction of other features** such as number of cells (**??**)
- **scene understanding** and **image interpretation** as components of high-level vision, no doubt the most difficult in this list and the least developed.

1.5 Computer vision, not an axiomatic science

Unlike, say, quantum mechanics, there is no single equation that governs computer vision, from the solution of which all else could be derived. Instead, we happen to live in a world that results from a few billion years of evolution under a small set of natural forces, from poorly understood initial conditions. In other words, our world could certainly look different from the way it does, and there is no deeper reason behind why a pedestrian looks the way she does. And yet, pedestrians need be detected, and the same goes for the other tasks mentioned above. This large diversity of the capabilities that we wish to cast into algorithms prevents image analysis from being an axiomatic science. This is of course bad news for anyone wishing for a logical, deductive structuring of this field; and it makes it difficult to bring the highly interrelated concepts and techniques relevant to the field into a linear order, such as required by a script; but even so, there is a set of deep realizations and aesthetical abstract concepts that make the field attractive also for the mathematically inclined, and it is simply great fun to play with abstract concepts and computers and to collaborate with experts from Engineering, the Natural Sciences and

Medicine on the analysis of the interesting data that they acquire. Not least, the effect of many operations can be directly visualized and image analysis is hence a kind of "visual mathematics". It is an interesting and rewarding field to work in, a field that we now finally take the first step towards.

1.6 Summary

- "Perception is not a clear window onto reality, but an actively constructed, meaningful model of the environment" [20].
- To the computer, all images are equally meaningless.
- The space of all images is so vast that naïve learning of a universal, say, object detection function in it is hopeless.
- Machine vision comprises a multitude of tasks that can roughly be grouped as image processing (low level), image analysis (intermediate level) and computer vision (high level analysis).
- No closed mathematical formulation of the computer vision problem exists; instead, it builds on a wide range of mathematical and algorithmic techniques.

1.7 Further reading

A readable introduction to animate vision including a good review of psychophysical experiments is offered by [20]. A brilliant collection of optical illusions can be found at [2]. "The Library of Babel" is a dazzling short story first published in the 1941 collection "El Jardín de senderos que se bifurcan" (The Garden of Forking Paths). English translations are available on the Internet.

Part I

The Basics of Learning

2 Introduction to classification

2.1 Overview

This chapter introduces a simple though fundamental classifier, along with some of the field's jargon. We will also see one of the many ways in which naïve machine vision lacks the robustness of its animate paragon, and what can be done about it.

2.2 Digit recognition

Let us begin with a modest example, digit recognition, as used in the automated dispatch of postal mail. Let us also assume that somebody has done all the hard work for us, viz. identified that part of an address field that contains digits, and decomposed this area into unique patches containing one digit each. Let us assume that each patch is of size 28×28 pixels. Our task is to automatically predict which of the digits $0, \ldots, 9$ a patch contains, or to flag a patch with "doubt" if we cannot make an assignment.

Now, if you knew nothing about or wanted to forego statistical learning, you could start and devise a set of rules expressing your prior knowledge. For example, you could look at the spread of the dark pixels in the horizontal direction: if it is small, chances are it could be a 1 (but perhaps also a 7). If a digit has two "holes", it could be an 8 (but perchance also a 0 with a flourish). Things quickly become complicated: think about how one would code criteria such as "a 3 has two arcs which are open to the left" in a real

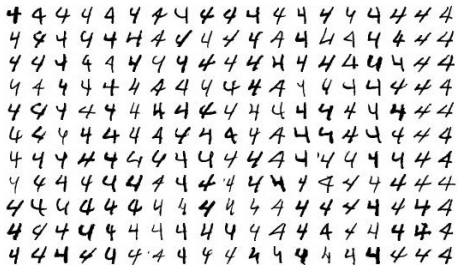


Figure 2.1: Some examples of class 4 from the famous MNIST (Modified National Institute of Standards [17]) database for isolated handwritten digit recognition.

programming language... It may just be feasible, but certainly is a difficult task.

As someone who is reading this script in the first place, you probably prefer to first collect a **t**raining set of examples, each of which is characterized by **f**eatures and a **l**abel. In digit recognition, the features could simply be the pixel intensity values in the patch, and the label would be the true digit. Why not make a prediction regarding an unknown patch by finding the most similar member of the training set, and using its label as a proxy? If this is your suggestion, then you have just invented one of the most fundamental and important algorithms in machine learning: the one-nearest neighbor classifier (1-NN)!

2.3 Nearest neighbor classification, and terminology

Classification, then, is the process of predicting the categorical label of an object from its features. **Categorical** means "discrete, with no implied ordering", and typical classification tasks are "intact vs. defect" in industrial quality control, "mitosis vs. meiosis" in quantitative biology, "positive vs. negative" in diagnostic settings, or "0 vs. 1 vs. ... vs. 9" in digit recognition. Two-class problems are also denoted as **dichotomous** or **binary** classification tasks.

In most cases, the features or **input** are assumed to be **flat** or **unstructured**, meaning they can be represented as a vector of numbers or categories. In this setting, each **pattern** or **observation** or example corresponds to one point in a **feature space** with as many dimensions as we have features.

In the digit recognition example introduced above, all the pixel values in a patch can be mapped in a fixed but arbitrary (customarily: lexicographic order) to the coefficients of a vector. As a consequence, each point in the training set, i.e. each image patch, becomes one point in feature space. Note that the choice of this particular feature space is arbitrary, because we might also have preferred to use features other than the raw pixel values, e.g. the spread of dark pixels in the horizontal direction, and many others. In more complicated examples than the one studied here, a proper choice of features becomes absolutely crucial, and without too much exaggeration one may say that conventional texts on image processing exclusively deal with **feature extraction**. We will return to this issue again and again, but shall postpone it for now.

So, once we have settled on an unstructured set of features, each training example becomes one point in feature space. We can now turn to the question of what is a good similarity measure in this space. Again, we have to brace ourselves for an unpleasant truth: there is no universally optimal similarity

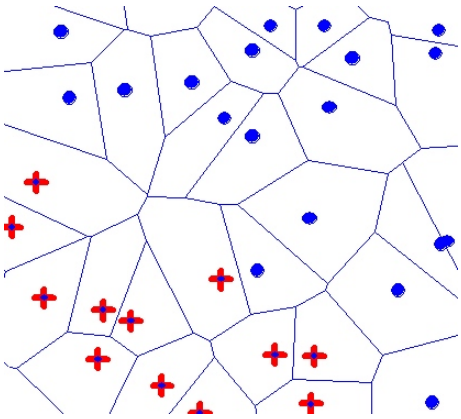


Figure 2.2: 1-nearest neighbor (1-NN) classification example: feature space is partitioned into Voronoi regions belonging to labeled training examples; to make a prediction, simply check which Voronoi region the query point falls in, and output the corresponding label.

measure. However, this blemish is also a blessing, because it is here that we can inject prior knowledge into the process, by formulating an appropriate similarity measure.

If no such prior knowledge is available, the most frequent choice is to use the $\mathbb{L}_2$ or Euclidean distance, which can be defined for two vectors $x_i, x_j \in \mathbb{R}^{\mathfrak{p}}$ of arbitrary dimensionality $\mathfrak{p}$ as

$$||x_i - x_j||_2 = \left(\sum_{k=1}^{\mathfrak{p}} \left([x_i]_k - [x_j]_k \right)^2 \right)^{1/2} \tag{2.1}$$

Given a similarity measure, the **1-nearest neighbor classifier (1-NN)** for an object of unknown class membership proceeds by comparing a query point with all the labeled points in the training set and using for prediction the label of that training point that is most similar. The 1-NN classifier effectively induces a partitioning or **tesselation** of feature space into non-overlapping regions each of which is associated with a single object from the training set. If the Euclidean norm is used as a dissimilarity measure, the resulting partitioning is a **Voronoi tesselation**[1], see Fig. 2.3

The 1-NN classifier immediately begs some generalizations: for instance, if there is some overlap of the different classes in feature space, the 1-NN classifier will create small "islands" of the wrong class within a sea of objects from the locally dominant class; these islands are an indication that indeed there *is* class overlap, and that classification in this area of feature space

[1]Being a fundamental concept in computational geometry, the Voronoi tesselation has been discovered multiple times in the context of different scientific fields, where a Voronoi zone or region is also known as, e.g., Wigner-Seitz cell or Brillouin zone or Dirichlet domain.

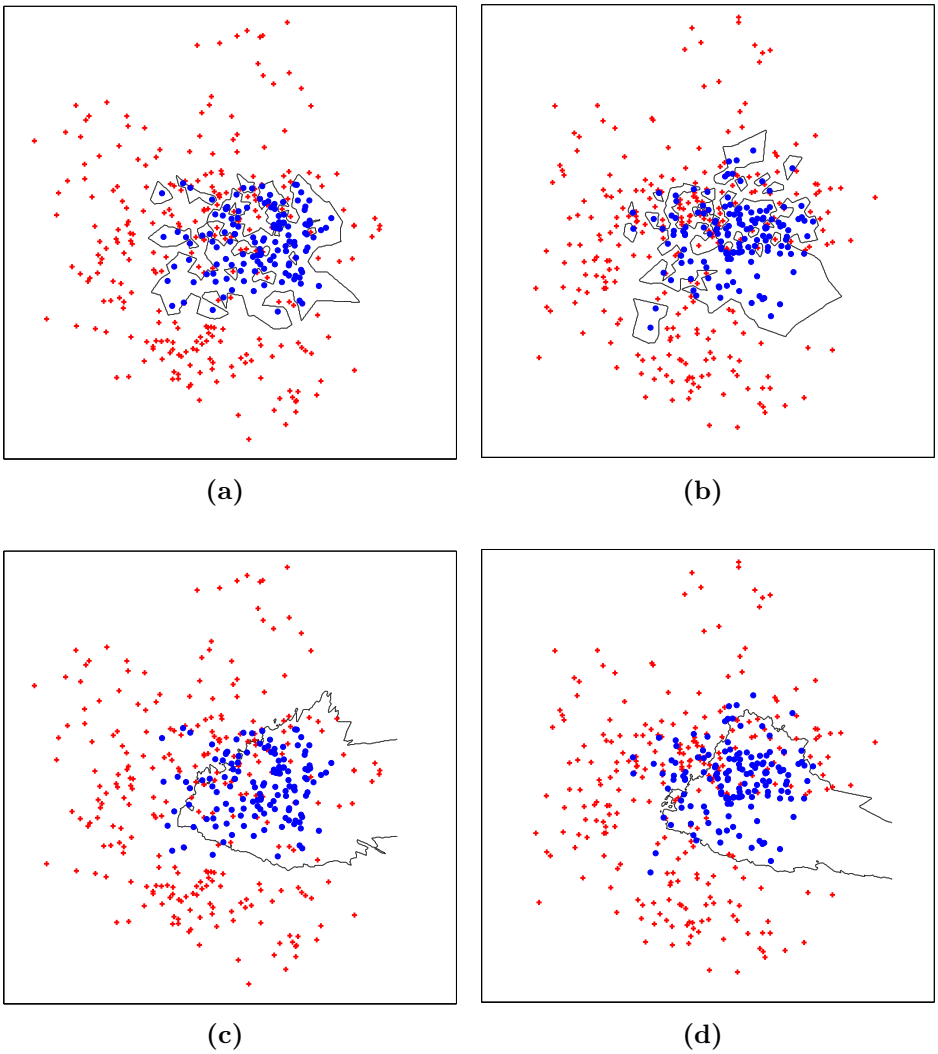


Figure 2.3: **(a)**, **(b)** 1-NN classifiers inferred from two disjunct halves of a single training set. Note the important and undesirable differences in regions where the two classes overlap. **(c)**, **(d)** 41-NN classifiers inferred from the same reduced training sets. These classifiers are more similar, at least in those regions of feature space where the density of observations is high.

will be error-prone; but the exact location of these islands will vary from one training set to the next and is inconsequential, see Fig. 2.3. One way forward is to search not for one, but for the k nearest neighbors – hence the name k-nearest neighbor classifier, or k-NN for short – and make a prediction based on which class is dominant among those k neighbors. Increasing k will lead to a **regularization** of our classifier, i.e. a "stabilization" with respect to random variability in the training data.

This sounds almost like a free lunch – why not increase k further and further? Carrying it to the extreme, reconsider Fig. 2.3: there are only about 120 points of the "blue disk" class. If we set k to a value larger than twice that number, we would *always* predict the "red cross" class, irrespective of where the query point lies in feature space. This reduces the nearest neighbor classifier to a mere majority vote over the entire training set, i.e. the available features are ignored completely. So far, we have looked at two extreme cases: choosing $k = 1$ or choosing k larger than twice the cardinality of the minority class in the training set; but what happens in between? A little more thought shows that choosing a large k will eliminate concave protrusions of the hypothesized ideal decision boundary (see next chapter for a derivation of this ideal boundary).

Summarizing our findings, choosing k too small will lead to a large random variability of the classifiers that are trained on different sets of observations from the same source – i.e., the classifiers will have a high **variance**. Choosing k too large will yield classifiers that are overly simplistic and ignore the more subtle features of the distribution of the classes in feature space – i.e., these classifiers will have a large **bias**. This so-called **bias-variance tradeoff** is paramount to data analysis, and we shall return to this point again and again in the following chapters.

Coming back to the toy example from Fig. 2.3, surely there must be an optimum choice of k? Yes there is, and in situations where there is an ample amount of data, it can be determined as follows. Take the set of all observations, and split it into three parts: a training set, a **validation set**, and a **test set**. The validation set can be used to see what parameter settings give the best results; while the test set is used to estimate the classifier's performance objectively before deploying it. The reason for the (important!) distinction between validation and test set is the following: especially in very flexible classifiers with a large number of parameters to tweak, repeated use of the validation set effectively means it is used for training. To make this very obvious, consider the following setting: an opportunistic binary "classifier" predicts a label for each observation in the validation set, and is told the number of errors it committed. It now submits the same predictions, except for a single observation where the prediction was changed; if the updated number of errors has increased, it will know that the previous label was correct, if the number of errors has decreased by one it will know that the new label is correct. Given enough training iterations, the classifier will perform perfectly on the validation set, but will have learned nothing about the true distribution of the classes in feature space.

Obvious as it may sound, this is a severe issue in real life: for many a researcher or engineer the temptation is too great and they keep changing their classifier until they obtain a fine performance on the test set; but in doing so they have **overtrained** a system which will later disappoint the

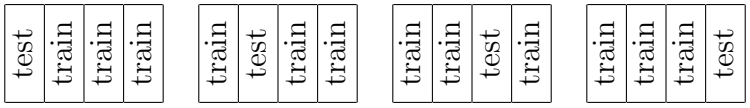


Figure 2.4: Schematic example of a four-fold cross validation. The data (features and labels) are split into four parts. All but one part or "fold" are iteratively used for training, and one for testing. In typical use, 10 or more "folds" are used.

collaborators or customers, or will frustrate the research community because the overly optimistic published results cannot be reproduced.

Alas, the splitting of all observations into three parts, the training, validation and test set, requires a generous abundance of labeled observations which in real life we are seldomly sufficiently fortunate to encounter. This is especially true if the data is high-dimensional, as will be discussed in section 4.2.1. In these settings, it becomes necessary to somehow "recycle" the observations or to otherwise prevent overtraining (statistical learning theory, see chapters **??**, **??**).

"Recycling" observations is at the core of **cross-validation (CV)**, see Fig. 2.4. The performance of a method can then be averaged over the testing results obtained, with the added benefit that some kind of confidence interval can be estimated [1].

2.3.1 k-NN classifiers at wits' end

With reference to the beginning of this chapter, we see that we have not yet delivered all the goods: we now are in a position to predict, e.g., the digit that an image patch shows; but as of now we have no mechanism to flag a patch with "doubt" if a new patch looks different from anything that was seen in the training set.

One heuristic is to define a threshold and compare the distance to the kth nearest neighbor to it: if that furthest of the neighbors that are used for prediction is closer than the threshold, we conclude that all is well and make a predicition based on a majority vote. If, on the other hand, that distance is larger than the threshold, then we may conclude that the query point is located in a region of feature space that is poorly covered with examples from the training set – in other words, the training set no longer is representative for the point in question, and a prediction would be pretty much random; in these cases, it is wiser and more honest to announce "doubt".

Another variant is the **ε-nearest neighbor classifier**: here, all training set observations within a sphere of radius ε around the query point are identified, and a majority vote is based on them. As above, "doubt" can be

expressed if the total number of observations within the prescribed radius is smaller than some user-defined threshold. Actually, more refined schemes are possible (**??**), where both the absolute number of points as well as the relative dominance of one class are taken into account.

Finally, one blemish of both the ε- and k-NN classifier is their hard cutoff: the nearest neighbors have equal weight in the prediction, irrespective of how close they are to the query point; whereas all remaining points have exactly zero weight. It seems more plausible to have a weight that decays with distance, and indeed such schemes have been developed, see chapters **??**, **??**.

2.3.2 Editing methods

In a sense, the k-NN classifier has zero training time (the training set only needs to be stored) and all the computational effort falls due at prediction or test time. This is in contrast to, say, neural networks whose training may take several weeks of CPU time, but where prediction is almost instantaneous (**??** and [17]).

Careful scrutiny of Fig. 2.3 reveals that only a fraction of the observations are actually involved in the definition of, or "support" the decision boundary: only those points belonging to different classes whose Voronoi regions share a face. How large this fraction is of course depends on the concrete training set. In general, it will be close to 1 if the data is high-dimensional or the classes have much overlap, but can also be very small if the data is not too high-dimensional and the classes are well-separated. When using 1-NN, these "insignificant" training points can be identified and simply be omitted without affecting the predictions at all. A number of editing methods have hence been devised [1] that seek to retain only the relevant points. While some of these methods aim at a mere speed-up without modifying the decision boundary [1], others seek to simultaneously introduce a regularization [1] by eliminating those training points that are dominated by representatives of another class. Note that this is very close in spirit to the "support vectors" of support vector machines, see **??**.

2.3.3 Computational issues

Since only the rank order of the distances matters, monotonic transformations of distances are without consequences. In particular, squared Euclidean distances can be used instead, obviating the unnecessary computation of the square root.

A Euclidean distance matrix is of course symmetric with 0s on the diagonal, so only the upper triangle need be computed; however, in matrix-oriented programming languages such as matlab it can be advantageous to ignore this

```
function d = squ_distance_matrix_of_point_sets(x1, x2)
% Computes squared distance matrix between two sets of points
% with p dimensions and n1 and n2 points, respectively.
%
% INPUT:
% x1     - required. (p by n1)-matrix of points
% x2     - required. (p by n2)-matrix of points
% OUTPUT:
% d      - is a (n2 by n1) matrix of squared Euclidean distances

x1sq = sum(x1.^2, 1);
x2sq = sum(x2.^2, 1);

n1 = size(x1, 2);
n2 = size(x2, 2);

d = repmat(x1sq, n2, 1) + repmat(x2sq', 1, n1) - 2*x2'*x1;
```

Figure 2.5: Fast matlab code for computing squared Euclidean distances between two sets of points.

symmetry and compute the full distance matrix, avoiding loops. The code snippet in Fig. 2.5 is based on the equality

$$||x_i - x_j||_2^2 = (x_i - x_j)^T(x_i - x_j) = x_i^T x_i - 2x_i^T x_j + x_j^T x_j \tag{2.2}$$

Finally, approximate nearest neighbor schemes can be used to alleviate the effort of a naive implementation which, with its cost of $\mathcal{O}(\mathfrak{n})$ for a single prediction based on $\mathfrak{n}$ points in the training set, can be excessive.

2.4 Re-enters image processing

The last few sections have completely ignored the digit recognition application that got our discussion started: all of what has been said holds in general and does not take the image character of the raw data into account. This crude approach is justified in the sense that it reaches a respectable performance: on the MNIST data, a 3-NN classifier reaches an error rate of about 5%, which is not bad for a 10-class problem. Not bad, but still 10 times worse than state-of-the-art results on that same set. To narrow this gap, we can make use of more powerful classifiers, as discussed in chapters **??**, **??**; and we should remember that we deal with image patches, and must ready ourselves to respect their peculiarities. It turns out that some of the idiosyncrasies of this data are not specific to digits, but pertain to all kinds of imagery.

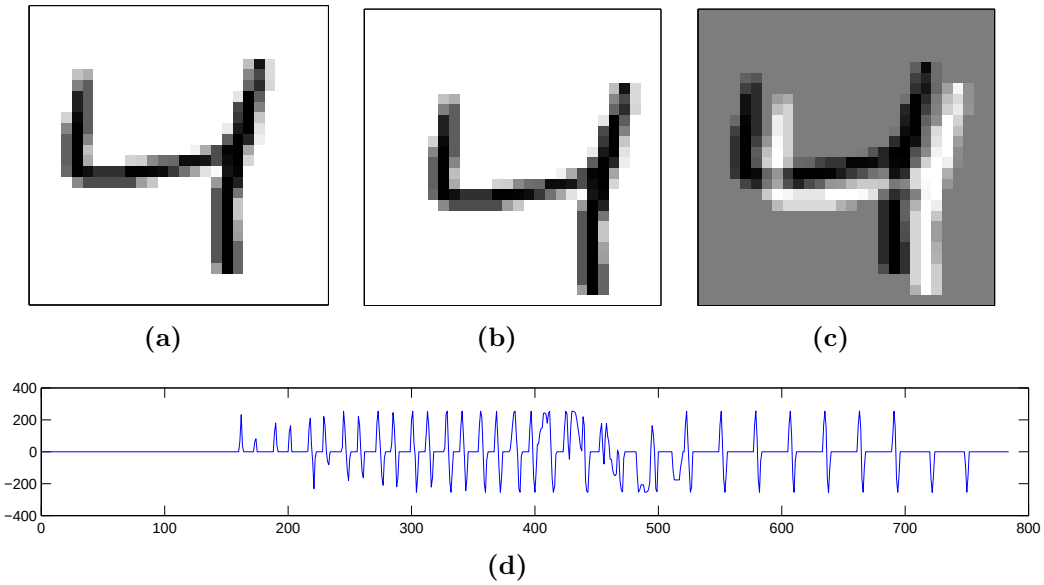


Figure 2.6: **(a)** A sample digit "4" from the MNIST training set, **(b)** a shifted version thereof, **(c)** difference image, **(d)** coefficients of vectorized difference image. To us, **(a)** and **(b)** are virtually the same, to the computer they are very different **(c)**,**(d)**.

Consider the Fig. 2.6(b), in which the digit has been shifted by a few pixels which to us is a trivial modification. To the computer, however, this shift leads to a completely different pattern, as evidenced by the large distance vector which is shown in patch format 2.6(c) and in terms of coefficients of the high-dimensional feature vector 2.6(d). It is important to understand that a linear translation in image space corresponds to a highly *nonlinear* trajectory in feature space, except if the features are translation-invariant by construction. When the pixel values themselves are used as features, such invariance is not given.

To fully understand this important point, consider an even simpler example: an "image" consisting of just three adjacent pixels. Let the first one be white and the other ones be black, i.e. the image is represented by the feature vector $(1, 0, 0)^T$. Now start to translate this image: after some time, the white pixel will have arrived in the middle and finally in the last pixel, corresponding to points $(0, 1, 0)^T$ and $(0, 0, 1)^T$ in feature space. The detailed trajectory in-between these three points will depend on the particular interpolation function used, but still these points will never lie on a straight line: in fact, the shifted image has "visited" all three coordinate axes of this primitive image space.

We can now generalize this finding. Consider in-plane rotation as another example of a simple rigid transform which is governed by just one parameter, the rotation angle. If we take an arbitray image of, say, the Eiffel tower and

gradually rotate it through 360°, in the feature space of all pixel values this image will trace out a complicated, but locally one-dimensional, closed curve: after one full rotation, we are back at the starting image / starting point.

In digit recognition, large rotations are not permissible (they would turn a 6 into a 9), but small rotations certainly do occur[2]. If we had an infinite training set, all these slightly rotated versions of sample digits would be represented and no further thought on our side would be required. As things are, the training set is finite and many of these expected variations are missing. Looking at Fig. 2.1, we see that beyond translation and rotation, other changes that should be taken into account are minor shearing and stretching, as well as the thickness of the pen used.

One brute-force possibility would be to enrich the training set with artificial examples that have been generated by affine operations (composed of some combination of translation, rotation, scaling and shearing) as well as morphological operations (see chapter **??**) to emulate different pen thickness or pressure applied. Indeed, this works well, but incurs a high computational cost [18]. A smart approximation is presented next.

2.4.1 Tangent distance

The set of all images that can be generated by modifying an original patch forms a **manifold** in feature space. The **nominal dimension** of this manifold is given by the dimensionality of the feature space itself; the **intrinsic dimension**, though, is given by the number of variables that are required to account for local displacements along the manifold. For a concrete example, consider once more a patch showing a digit. As described above, rotation alone will trace out a *locally* one-dimensional curve, or manifold, in feature space that forms a loop if we rotate through 360°. Magnification or shrinking of the original picture will lead to another nonlinear curve which does not intersect with or close on itself. If we now consider all possible rotations and, say, all magnifications or shrinkages up to a factor of two (while keeping the number of pixels constant), this corresponds to a twisted band in feature space; topologically speaking, this band is equivalent to a frisbee, so only two coordinates – rotation and scaling factor – are required to uniquely determine the position in feature space: the intrinsic dimension is merely two, while the nominal dimension for an MNIST digit with $28 \times 28 = 784$ pixels is 784.

In the **tangent distance** algorithm [23], the exact nonlinear manifold induced by a set of transformations is approximated by a linear manifold, or hyperplane, around the original patch. These planes will, in general, never intersect because they are of much lower dimensionality than the feature space (for illustration, consider two randomly oriented straight lines in $3D$

[2] Translation itself has been eliminated in the MNIST data set by subtracting the spatial "center of mass" of the gray values.

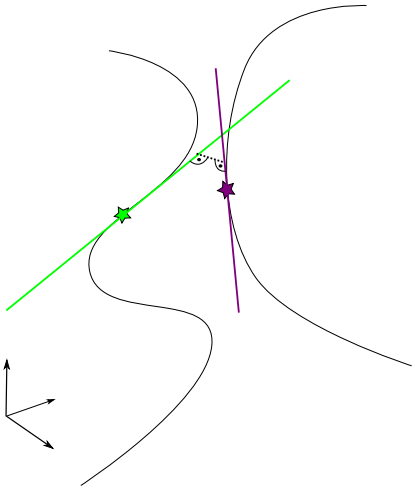


Figure 2.7: Cartoon illustration of tangent distance. In feature space, the original patterns (image patches) are marked by stars. Modifications of these patches such as translation, rotation, etc. lead to complex nonlinear displacements along low-dimensional manifolds in feature space (black curves). The manifolds are locally approximated using tangents which generally will *not* intersect (feature space is high-dimensional). The closest Euclidean distance between the two tangents (indicated by dashed line) is the "tangent distance".

and the vanishing probability that they intersect). Even so, two hyperplanes will always have a point of closest approach, and it is this closest distance that is defined as the tangent distance, see Fig. 2.7. The prime motivation for this algorithm is computational efficiency: it is *much* cheaper to compute the tangents and find the shortest distance between these (by solving a linear least squares problem), than to trace out the complete nonlinear manifolds and find their shortest distance. Simard et al. [23] discuss additional means of improving the approximation (by iterating tangent distance computations) and, in particular, how to further speed up the calculation by means of approximate early stopping criteria.

On the MNIST data, using tangent rather than Euclidean distance in conjunction with k-NN reduces the error rate by about 50% – this great improvement attests to the importance of including prior knowledge when available. In the case discussed here, it is the desired **invariance** of the pattern recognition with respect to minor translations, rotations, scaling, etc. that is built

into the system.

Note that we have elegantly glossed over the all-important "detail" how a tangent is actually computed; or, in the exact (manifold rather than tangent distance) algorithm, how a rotated or scaled version of an image can be obtained. These questions boil down to the problem of how to properly interpolate a discrete image. This issue is central to image processing and, as in our example, pattern recognition. We will turn to this problem in chapter ??.

2.5 Summary

- k-nearest neighbors (k-NN) is a simple and well-understood classifier with respectable performance.
- The number of k nearest neighbors that are considered for a prediction trades off bias and variance of the prediction.
- Cross-validation (CV) can be used to find the optimal tradeoff.
- Unless a suitable (i.e.: invariant) representation is used, simple transformations in image space may lead to complicated changes in feature space.

3 Some theory behind classification

3.1 Overview

The previous chapter has vaguely referred to an "ideal" classifier. In this chapter, we will see the good news that such an ideal classifier indeed exists and can be formalized – and the bad news that it is not attainable in practice. We then encounter linear and quadratic discriminant analysis, classifiers which in many ways are complementary to k-NN, and study their merits and limitations.

The price for the good news is that treatment will of necessity be more formal in this chapter. However, this level of treatment is indispensable to access recent literature, and to carry the arguments beyond cocktail party standards. So do not be discouraged, and plow through...

3.2 Statistical Learning Theory

In section 2.3 we postulated, in passing, that there is such a thing as an ideal decision boundary for a classification problem. To derive it, let us assume that the examples (x_i, y_i) in our training set have been sampled randomly, independently from each other from the same underlying distribution[1] which is assumed to be constant over time. The last assumption is generally extended to imply that future samples will come from the same distribution, i.e. the classifier will only be confronted with samples from the same distribution after deployment. This is a bad assumption in the real world where processes can drift, sensors can age, etc. meaning that the distribution does indeed change over time.

Assume, furthermore, that we have a classifier $f(x)$ that can predict a class or label $\hat{y}_i$, including the class "doubt", for each position x in feature space.

The final ingredient we need is a **loss function** $\mathfrak{L}(y, \hat{y})$ that tells us what loss we incur if the true label is y and our prediction is $\hat{y}$.

We can now compute the **risk** $\mathfrak{R}(f)$, that is the expected loss that depends on both the classifier f and the joint distribution of features and labels $(\mathbf{x}, \mathbf{y})$

[1] Statisticians write **iid** for short, i.e. independently and identically distributed.

in our data:

$$\begin{aligned} \mathfrak{R}(f) &= \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{y}}\mathfrak{L}(\mathbf{y}, f(\mathbf{x})) && (3.1)\\ &= \int_{\mathcal{X}} \mathbb{E}_{\mathbf{y}}\mathfrak{L}(\mathbf{y}, f(x))p(x)dx && (3.2) \end{aligned}$$

Note that both features and labels are now treated as random variables, hence the bold print (see page 61 for an overview of the notation used, and section **??** for an introduction to random variables). In the last equation, we have assumed that feature space is continuous, $\mathcal{X} \subset \mathbb{R}^{\mathfrak{p}}$.

Our aim is to find the classifier that minimizes this risk. The first expectation goes over all of feature space; to minimize the overall risk, we need to minimize it, individually, at each position of feature space. Let us hence concentrate on the second expectation,

$$\mathbb{E}_{\mathbf{y}}\mathfrak{L}(\mathbf{y}, f(x)) = \sum_{y\in\mathcal{Y}}\sum_{z\in\mathcal{Y}} \mathfrak{L}(y, z)\mathcal{I}(f(x) = z)\mathbb{P}(\mathbf{y} = y|x) \tag{3.3}$$

In words, eq. 3.3 says the following: we need to sum over all the possible discrete labels $y \in \mathcal{Y}$ that we *could* observe at location x in feature space with probability $\mathbb{P}(\mathbf{y} = y|x)$ (the latter is the conditional probability that random variable $\mathbf{y}$ assumes the value y, given position x: see appendix **??**). The classifier $f(x)$ itself is deterministic, it predicts label z, thus incurring a loss of $\mathfrak{L}(y, z)$. Since we do not know in advance which label z the classifier will choose, we need to sum over all the possibilities $z \in \mathcal{Y}$, but only pick the one loss that corresponds to the classifier's choice. This is achieved by means of the **indicator function** $\mathcal{I}$ that is 1 when its argument is true, and 0 otherwise. $\mathcal{Y}$ is here assumed to contain the "doubt" class; if all training observations are labeled, then their $\mathbb{P}(\mathbf{y} = \text{doubt}|x)$ is zero, but the classifier is still permitted to express doubt.

If we make the reasonable assumption that correct predictions incur no penalty, $\mathfrak{L}(y, y) = 0$, eq. 3.3 can be rewritten as

$$\mathbb{E}_{\mathbf{y}}\mathfrak{L}(\mathbf{y}, f(x)) = \sum_{y\in\mathcal{Y}\setminus f(x)} \mathfrak{L}(y, f(x))\mathbb{P}(\mathbf{y} = y|x) \tag{3.4}$$

The contents of the loss matrix should reflect the severity of the different possible errors. For instance, in a diagnostic setting, the costs of a false negative (in the extreme case, the life of a patient that remained untreated) are much higher than the costs of a false positive (in diagnostic settings, typically the cost of a second, independent test to confirm or reject the first finding).

As an important special case, let us consider the following loss matrix:

$\mathfrak{L}(y, f(x))$	$f(x) = 1$	$f(x) = 2$	$\dots$	$f(x) = \mathfrak{c}$	$f(x) = \text{doubt}$
$y = 1$	0	$\mathcal{P}_m$	$\dots$	$\mathcal{P}_m$	$\mathcal{P}_d$
$y = 2$	$\mathcal{P}_m$	0	$\dots$	$\mathcal{P}_m$	$\mathcal{P}_d$
$\vdots$	$\vdots$		$\ddots$		$\vdots$
$y = \mathfrak{c}$	$\mathcal{P}_m$	$\mathcal{P}_m$	$\dots$	0	$\mathcal{P}_d$

It suggests a constant penalty for misclassification $\mathcal{P}_m$ as well as a penalty $\mathcal{P}_d$ when the classifier resorts to announcing "doubt" instead of making a prediction. Using this symmetric loss function and remembering that conditional probabilities must sum up to one, $\sum_{y \in \mathcal{Y}} \mathbb{P}(\mathbf{y} = y|x) = 1$, eq. 3.4 reduces to

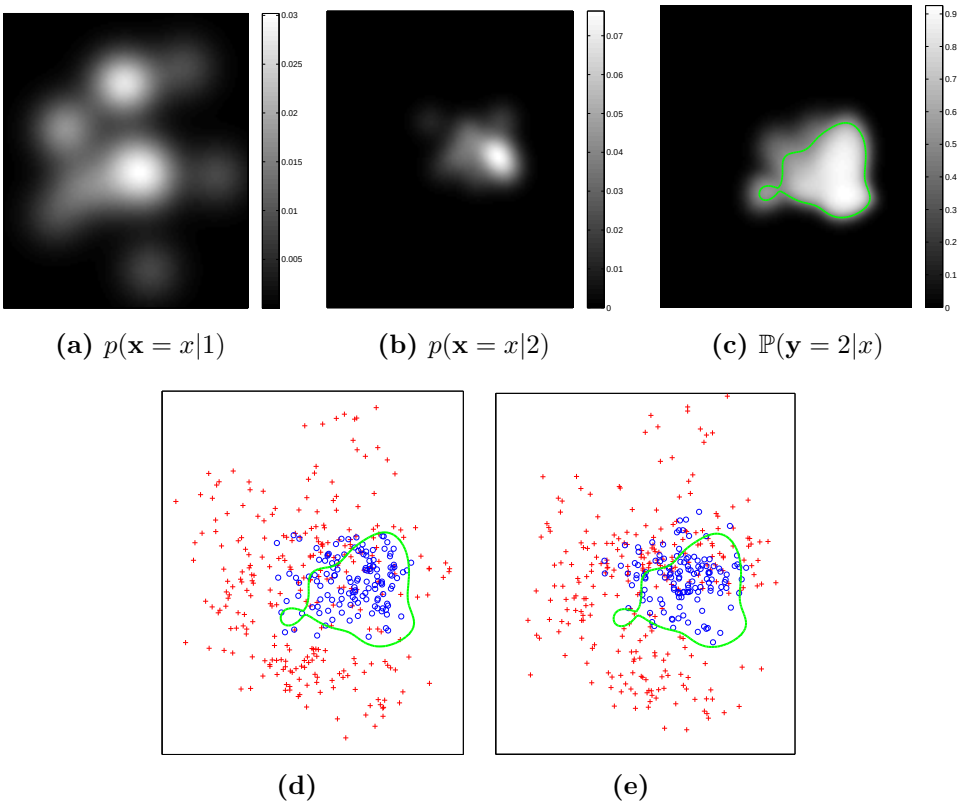
$$\begin{aligned} \mathbb{E}_{\mathbf{y}} \mathfrak{L}(\mathbf{y}, f(x)) &= \mathcal{I}(f(x) \in \{1, \dots, \mathfrak{c}\})(1 - \mathbb{P}(\mathbf{y} = f(x)|x)) \times \mathcal{P}_m \\ &\quad + \mathcal{I}(f(x) = \text{doubt}) \times \mathcal{P}_d \end{aligned} \tag{3.5}$$

In words, if the classifier announces "doubt", the cost incurred is always $\mathcal{P}_d$; if it does predict one of the $\mathfrak{c} := |\mathcal{Y}|$ classes $1, \dots, \mathfrak{c}$, the cost is $\mathcal{P}_m$ times the probability of observing a class other than the predicted one at position x in feature space.

The *optimal classification strategy* is now clear: to minimize the risk, the classifier should predict the most abundant among the classes $1, \dots, \mathfrak{c}$ to minimize the probability of misclassification. But overall, the cost of a misclassification $\mathcal{P}_m$ times the probability of an error should be smaller than the cost of announcing doubt $\mathcal{P}_d$; if it is not, it becomes "cheaper" for the classifier to remain noncommittal and simply express "doubt". In other words, if the user-specified cost for doubt $\mathcal{P}_d$ is low, the classifier will only make predictions in cases in which it is absolutely sure. If $\mathcal{P}_d \gg \mathcal{P}_m$, the classifier will always make a prediction, even if no class clearly dominates the others at that point in feature space.

The classifier we have just derived is the best classifier there can be; it is called the **Bayes classifier**. Among all strategies, it is the one that achieves the minimum possible risk, which for a given distribution of classes is called the **Bayes risk**. The Bayes risk depends on how the different classes are distributed in feature space: if they have a lot of overlap, the Bayes risk will be large (meaning that even this very best classifier will necessarily have a large error rate); whereas if the classes are well-separated, the expected loss will be low. This again underscores the importance of finding good features for a problem at hand. In the sense of classification, the hallmark of good features is a clear separation of the classes in feature space, and consequently a low Bayes risk.

Overall, we come to the reassuring realization that this optimum classifier which we have obtained in a lengthy derivation actually has common sense: at query point x in feature space, it simply predicts that class y which has the highest posterior probability $\mathbb{P}(y|x)$ at this point in feature space. That is, it chooses the locally dominant class, or announces doubt if there is no clear domination of a single class.



(a) $p(\mathbf{x} = x|1)$ **(b)** $p(\mathbf{x} = x|2)$ **(c)** $\mathbb{P}(\mathbf{y} = 2|x)$

(d) **(e)**

Figure 3.1: **(a)** Density of class 1, **(b)** of class 2 and **(c)** posterior probability of class 2. The Bayes classifier decision boundary (green line) for a binary classification problem is given by the set of points at which $\mathbb{P}(\mathbf{y} = 2|x) = 1/2$. **(d)**, **(e)** Two training sets drawn from the above true distribution, with the Bayes decision boundary overlaid. These samples illustrate how difficult it is to precisely estimate the Bayes classifier given a finite training set.

3.3 Discriminative vs. generative learning

So far so good, but why did we discuss k-nearest neighbors in the first place, if the ideal classifier is known and that simple? The answer is, of course, that the formulæ above are based on knowledge of the *true* $\mathbb{P}(y|x)$, which is not available – all we have is a finite training set! By looking at Fig. 3.1 we can see just how difficult it is to estimate the Bayes classifier given a finite training set.

In retrospect, then, k-NN appears in a different light: it can be seen as an *empirical* estimate of the true posterior probability in a local environment around the query point.

Given Bayes' theorem (see appendix **??**)

$$\mathbb{P}(y|x) = \frac{p(x|y)\mathbb{P}(y)}{p(x)} \tag{3.6}$$

we see that there are two ways to approximate the Bayes classifier: by either modeling the left-hand side, i.e. directly estimating the **posterior probabilities** of the different classes; or by estimating the **class densities** $p(x|y)$ along with the **prior class probability** $\mathbb{P}(y)$. The former class of methods is called **discriminative** (because they directly try to distinguish the classes) whereas the latter are denoted generative. **Generative models** earn their name because they estimate the class densities – given these, it would be possible to simulate, or generate, new observations that approximately follow the same distribution as the ones in the training set.

In the light of the previous section, *a*ny method that can be used to estimate the density of a class in feature space is a potential building block for a generative classifier. A large number of density estimators exist, ranging from so-called **nonparametric** that make few assumptions on the distribution of the data (such as histograms, or kernel density estimates (section 4.2.1)) to **parametric** techniques that make far-reaching, if sometimes ill-justified, assumptions regarding the distribution from which the data have supposedly been sampled. Examples for this class are weighted sums of Gaussian distributions, so-called Gaussian mixture models (**??**).

The distinction between parametric and nonparametric is somewhat arbitrary, and the terms themselves are misleading: "parametric" models tend to have less flexibility, fewer degrees of freedom and fewer parameters; while the "nonparametric" methods are actually those with the largest, though sometimes implicit, number of parameters. Consider k-NN as an example: at first sight, it might seem like the method has only one adjustable parameter, namely k. However, this is a hyperparameter which adjusts the flexibility of the method. If we set k to 1, the true, implicit, number of parameters is upper bounded by the size of the training set! To see this, remember that the training data partition feature space into Voronoi regions; now, for heavily overlapping classes, each of these regions requires one (implicit) parameter: its label.

3.4 Linear discriminant analysis (LDA)

1-NN is a very flexible method that makes few assumptions indeed regarding the true distribution of the classes. As a "contrast program", we now turn to a method that is the direct opposite: linear discriminant analysis. It makes very strong (which means, unless they are verified: poor!) assumptions regarding the distribution of the classes in feature space: it assumes that each class has

a multivariate Gaussian (see appendix **??**) distribution

$$p(x|\mathbf{y}=i) \quad = \quad \frac{1}{(2\pi)^{\mathfrak{p}/2}|\Sigma_i|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T\Sigma_i^{-1}(x-\mu_i)\right) \tag{3.7}$$

with the *same* covariance function $\Sigma_1 = \Sigma_2 = \Sigma$, but different mean.

Given this crucial assumption, we can now set to work with the machinery from section 3.2 to develop a binary classifier. First, let us find the decision boundary between classes 1 and 2: it is given by that set of points at which both classes have the same posterior density. Solving for x by canceling terms and rearranging we find

$$\begin{aligned} \mathbb{P}(\mathbf{y}=1|x) &= \mathbb{P}(\mathbf{y}=2|x) \\ p(x|\mathbf{y}=1)\mathbb{P}(\mathbf{y}=1) &= p(x|\mathbf{y}=2)\mathbb{P}(\mathbf{y}=2) \\ 2\log\left(\frac{\mathbb{P}(\mathbf{y}=1)\,|\Sigma_2|^{1/2}}{\mathbb{P}(\mathbf{y}=2)\,|\Sigma_1|^{1/2}}\right) &= x^T\left(\Sigma_1^{-1}-\Sigma_2^{-1}\right)x \\ &\quad +x^T\left(-\Sigma_1^{-1}\mu_1+\Sigma_2^{-1}\mu_2\right) \\ &\quad +\mu_1^T\Sigma_1^{-1}\mu_1-\mu_2^T\Sigma_2^{-1}\mu_2 \end{aligned} \tag{3.8}$$

By the LDA assumption, $\Sigma_1 = \Sigma_2 = \Sigma$, so that

$$x^T\Sigma^{-1}(\mu_2-\mu_1)+\mu_1^T\Sigma^{-1}\mu_1-\mu_2^T\Sigma^{-1}\mu_2+2\log\left(\frac{\mathbb{P}(\mathbf{y}=2)}{\mathbb{P}(\mathbf{y}=1)}\right)=0 \tag{3.9}$$

This equation is of the form $x^Tw + b = 0$, i.e. it defines a (hyper-) plane with normal vector $w = \Sigma^{-1}(\mu_2 - \mu_1)$. It also explains the name of *linear* discriminant analysis: whatever the covariance function Σ that the classes have, the decision boundary is always a (hyper-) plane. If the classes are assumed to have an isotropic covariance function $\Sigma = \sigma^2 I$, the normal of this decision boundary is simply given by the difference vector of the two class means. The log ratio of the prior class probabilities simply shifts this decision boundary along the normal: if it is more probable to obtain a datum from one class than from the other class, then this shifts the decision boundary accordingly, to the profit of the more abundant class.

In practice, both the class means μ_i and the **precision function** Σ^{-1} have to be estimated from the training set, and both will be subject to random error. Fig. 3.2(c) illustrates that the classifier will be very confident in the upper right and lower left corner of the input domain even though there are no training samples in those regions. It is hence advisable to put a threshold on the density of the locally most abundant class: if the density falls below a user-selected value, it is a good idea to let the classifier express "doubt".

LDA tends to work well if only few observations are available in a high-dimensional feature space; but it clearly does not satisfy the requirements of the data set shown in Fig. 3.2 which calls for a nonlinear decision boundary. We can now relax one of the assumptions that LDA makes.

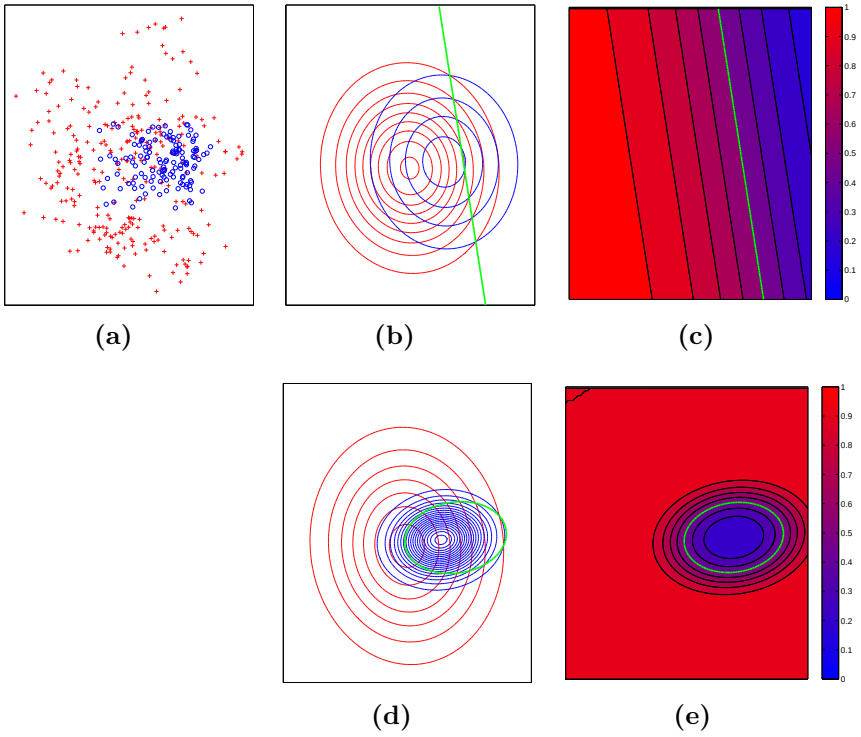


Figure 3.2: Linear (top) vs. quadratic (bottom) discriminant analysis. **(a)** The data set already familiar from Fig. 3.1. **(b)**, **(d)** Gaussians with identical and different covariance matrices fit to both classes and weighted with the empirical prior. The decision boundary is indicated in green. By definition, it passes where the first, second, ... contours from both classes intersect. **(c)**, **(e)** The estimated posterior probability of the "red" class.

3.5 Quadratic discriminant analysis (QDA)

We can again invoke eq. 3.8, but this time refrain from assuming that the covariance matrices Σ_1 and Σ_2 are identical. Eq. 3.8 now defines a **quadric**, i.e., depending on the parameters, a parabola or hyperbola or ellipsoid or pair of lines. Using two Gaussians with distinct parameters does the toy example in Fig. 3.2 more justice: the ellipsis is actually pretty close to the Bayes classifier, especially in the high-density region.

If no restrictions are imposed, the number of parameters that need to be estimated grows quickly with the nominal dimensionality $\mathfrak{p}$. At first glance, it looks like even LDA requires $\mathfrak{p}(\mathfrak{p}-1)/2$ parameters for the precision matrix Σ^{-1}, plus $2\mathfrak{p}$ parameters for the means μ_1, μ_2. However, remembering that eq. 3.9 is of the form $x^T w + b = 0$ shows that the effective number of parameters that need to be estimated is only $\mathfrak{p}+1$, for w and b. A similar argument shows that the effective number of parameters for QDA is $\mathfrak{p}(\mathfrak{p}-1)/2+\mathfrak{p}+1$.

For a feature space with high dimensionality $\mathfrak{p}$, there are generally not enough training samples available to estimate the parameters reliably. Jerome Friedman has shown that the so-called plug-in estimates are biased in the small sample case, and has proposed **regularized discriminant analysis** [7, 14, chapter 4.3]. It introduces two parameters α, β that reduce the flexibility of the resulting classifier. The first one "mixes" some of the estimated[2] pooled covariance matrix $\hat{\Sigma} = \sum_{i=1}^{\mathfrak{c}} \hat{\Sigma}_i / \mathfrak{n}_i$ (where $\mathfrak{n}_i$ is the number of training examples in class i) into the individual class covariance matrices $\hat{\Sigma}_i$:

$$\hat{\Sigma}_i(\alpha) = \alpha \hat{\Sigma}_i + (1 - \alpha)\hat{\Sigma} \tag{3.10}$$

The rationale here is that the pooled covariance matrix can be estimated more reliably, given that it is computed from *all* observations. The resulting classification boundary is hence simplified by making it a "mixture" of the full quadric and the linear classifier. The second parameter β biases the covariance matrix of each class towards isotropy:

$$\hat{\Sigma}_i(\alpha, \beta) = \beta \hat{\Sigma}_i(\alpha) + (1 - \beta) \frac{\operatorname{tr}\left(\hat{\Sigma}_i(\alpha)\right)}{\mathfrak{p}} I^{\mathfrak{p}} \tag{3.11}$$

When α, β are set to zero, the "nearest means" classifier results, which simply assigns each observation to that class with closest center of mass.

The parameters α, β can be estimated by "cross-validation", see 17. More recent work focuses on directly making the coefficients of w smooth or sparse, see **??**.

3.6 Summary

This chapter has been a little more difficult than the previous ones; but it has endowed us with some beautiful concepts. First of all, we now know what the perfect classifier looks like, and now better understand the fundamental limitations of *all* statistical classifiers: if the classes overlap in feature space, we cannot reduce our error rate below a given level dictated by the Bayes classifier. If this level is inadequate, we must necessarily add new, more informative, features. These will typically come at a price, for instance they may require the acquisition of new measurements with an extra sensor.

If all prediction errors are equally undesirable, the Bayes classifier reduces to predicting that class which has the highest posterior probability at a given point in feature space.

Linear and quadratic discriminant analysis are parametric classifiers that are useful in a wide range of problems; regularization techniques are available for both.

[2] Σ now carries a hat "ˆ" to stress that it has been estimated, and is not known exactly, cf. appendix **??**

All of this chapter was based on the assumption that the distribution in the training set is representative for samples that call for classification in the future. This is often not fulfilled if there is process drift, making the detection of outliers (**??**) and change points an important topic. Some theory is available for this so-called **covariate shift**, e.g. [24].

4 Introduction to unsupervised learning

4.1 Overview

In the previous chapters, we have cast image analysis as a pattern recognition problem, with each patch, or even complete image, corresponding to one point in feature space. We shall now look at ways of describing the distribution of these points in space. One natural way to do so is in terms of their density (section 4.2) which in turn allows us to group together similar points in clusters (section 4.3). Such a grouping can be seen as a form of lossy compression. A canonical way of finding the most relevant subspace, and hence also of compressing, the data is described in section 4.4. All of these methods assume there are no labels, or disregard them, hence the name of this chapter.

4.2 Density estimation

Because estimating the density of points is an operation of such fundamental importance, we bestow upon it the privilege of opening this chapter. Its applications are manifold: for instance, in a generative classifier, nonparametric density estimation allows us to relax the harsh assumptions of LDA and QDA (section 3.4, 3.5) which modeled the density of each class using a single multivariate Gaussian only. Density estimation can also be used for **outlier detection** in a classification problem: whenever a query point lies in a part of feature space that is not adequately covered by the training set, an alarm should be raised instead of making an uninformed prediction with all its possible adverse consequences. Density estimates are also used in nonparametric regression (**??**), for edge-preserving image smoothing (**??**) and for image segmentation (**??**). Finally, points that contribute to the same local maximum of the density may be considered to somehow belong together, a notion discussed in more detail in section 4.3.

Density estimation can also be cast as a supervised learning problem, by introducing an auxiliary class with known distribution (chapter **??**). However, the most typical approaches are variants of either histograms or kernel density estimation, which we turn to now.

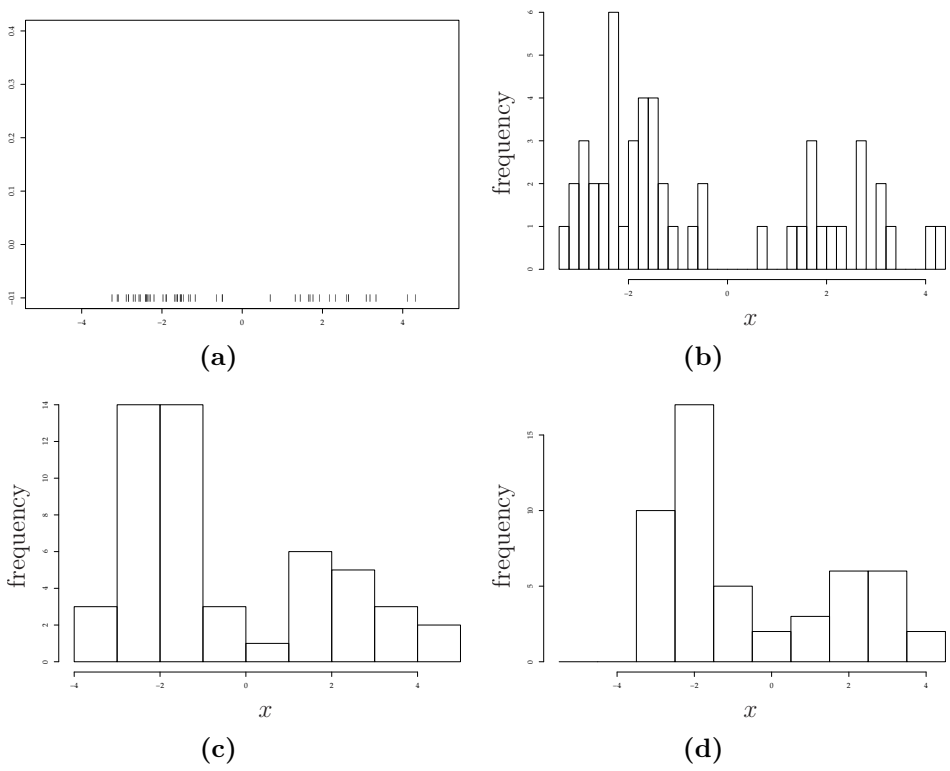


Figure 4.1: A set of one-dimensional observations and histograms thereof. Note the differences in appearance resulting from different parameter choices; in particular, **(c)**, **(d)** differ only in the offset of their bins.

4.2.1 Histograms and the Curse of Dimensionality

The best known density estimate is the **histogram**, which simply counts the number of occurrences in a set of prespecified bins. In most cases, the bins are defined by the cells of a lattice [6], i.e. by a set of identical but shifted volumes that tile space. One-dimensional regular histograms have just two parameters, the offset of the first and the width of all bins. Choosing bins that are too small will lead to histograms in which most bins are empty and many hold only a single sample; choosing a bin width that is too large may gloss over interesting structure in the data. As opposed to the supervised learning techniques discussed before, there is no mechanism to find the "true" bin width: what is an optimal choice will of course depend on the amount of data (with more observations permitting use of smaller bin widths and hence a finer rendering of details), but also on the structure of the data and, finally, the intent of the data analyst. This state of matters is of course unsatisfactory, but cannot be helped.

A generalization of the histogram to multiple dimensions is straightfor-

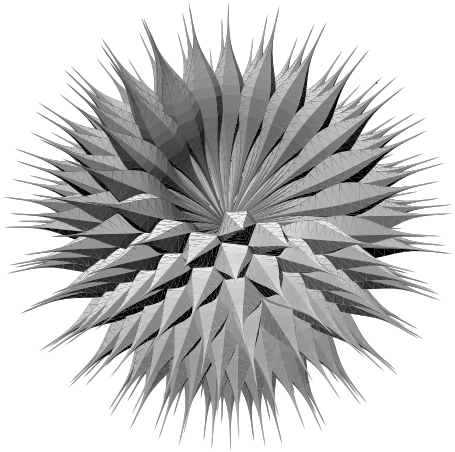


Figure 4.2: Three-dimensional sketch of an eight-dimensional hypercube which preserves the radial mass distribution and number of vertices. Some of the $2^8 = 256$ vertices have been broken away to reveal that both the center and the periphery of this cube contribute little to its overall mass [13].
(Courtesy of Erik Agrell.)

ward, but requires choice of a few more parameters: on the one hand, the precise rotation and shift of the first bin; and the shape of the bins. The consequences of parameter choices are aggravated when dealing with higher-dimensional data. In particular, the (hyper-) rectangular bins that are mostly used for reasons of convenience turn out to be a particularly poor choice in higher dimensions [13, 16]! This finding is related to a host of evidence that is commonly subsumed as **curse of dimensionality**. This name seeks to convey that our spatial intuition, trained in three dimensions, is ill equipped to understand the geometry of higher-dimensional spaces. As an illustration, consider the sketch in Fig. 4.2.

One manifestation of this "curse" is that in high-dimensional spheres, it is the "onion layer" just below the surface that contributes most to the overall mass. This is due to the volume element of integration in a spherical coordinate system, which grows as a monomial of the radius.

In a high-dimensional Gaussian distribution, this dominance of the outer layers is eventually offset by the probability density itself which decays exponentially with the square of the radius. The length r of a $\mathfrak{p}$-dimensional standard normally distributed random vector has a **chi distribution** with $\mathfrak{p}$ degrees of freedom, the probability density function of which carries the vestiges of these antagonistic effects: it is the normalized product of a radius monomial and Gaussian-type decay, with the non-negative real numbers as

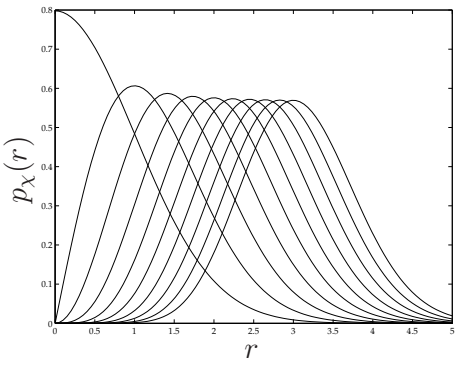


Figure 4.3: Probability density of the chi distribution with $1, 2, \ldots, 10$ degrees of freedom.

its domain:

$$p_\chi(r) = \frac{2^{1-\mathfrak{p}/2}}{\Gamma(\mathfrak{p}/2)} r^{\mathfrak{p}-1} e^{-r^2/2}, \quad r \in \mathbb{R}_0^+ \tag{4.1}$$

The **mode**, or maximum, of the distribution shifts with the dimensionality $\mathfrak{p}$ as $\sqrt{\mathfrak{p}-1}$. Interestingly, the variance quickly converges to a constant value with increasing dimensionality meaning that a normally distributed example is most likely to be found at a distance $\sqrt{\mathfrak{p}-1}$ from the origin, where the relative precision of this estimate increases with growing dimensionality.

An intriguing corollary is that the distance between any two points that are randomly sampled from a normal distribution *also* has a fairly narrow spread in higher dimensions. To see this, remember that the difference between two normally distributed random variables is also a Gaussian random variable, with a variance that is the sum of the individual variances (appendix **??**). Applying this argument to each coefficient of the difference vector shows that it, too, has a multivariate Gaussian distribution and that, consequently, the length of this difference vector follows a (scaled) chi distribution.

This finding carries over to other distributions with high intrinsic (section 2.4.1) dimensionality: empirically, the ratio of the greatest to the smallest distance between any two points from a finite set converges to one, as demonstrated also by simulation [3].

All of this implies that density estimation in high dimensions is an ill-posed problem; the number of points required for a reliable estimate grows at least exponentially with the dimensionality. For histograms, this means that if the data has high intrinsic dimension, choosing an appropriate bin size is difficult: as described above, most bins will have zero or one sample only if the bins are too small, or few bins will hold most of the data if the bins are too large. Things do not look as bad if the intrinsic dimensionality of the data is low; however in those cases, a mechanism to project the data to the relevant low-dimensional space is required. Such a mechanism is the subject

of section 4.4.

Average shifted histograms (ASH) have been proposed [21] as a remedy for the arbitrariness of the shift of the first bin. In ASH, one computes multiple histograms for different offsets, and averages over these to obtain a density estimate at a given location. From the perspective of the query point, this amounts to computing a weighted sum over nearby points. A moment's thought shows that the weight function must be a triangle in the one-dimensional case that has a weight of one at the query point and reaches zero at a distance of plus or minus the bin width. More precisely, for an infinite number of randomly shifted histograms, the ASH converges to

$$\mathrm{ASH}(x) = \int k(x-h) \sum_{i=1}^{\mathfrak{n}} \delta(h - x_i)\, dh$$

that is, a convolution (**??**) of the kernel function k with a **spike train**, or set of delta functions, that are located at the observations. The kernel is given by the autocorrelation (**??**) of the bin with itself. This formula also holds for multiple dimensions. (Average shifted) histograms are an important ingredient of the pyramid match kernel ([10], **??**), a popular method to compare sets of features, e.g. for object recognition.

If one were to generalize ASH to also include an averaging over possible bin widths, yet different and more complicated weighting functions would arise. Instead, one can choose these directly, resulting in the method described next.

4.2.2 Kernel density estimation (kDE)

In kDE, the user can directly pick a **kernel** or weighting function k. This requires specification of its shape (generally assumed to be isotropic, $k(x) = k(||x||_2)$, which does not limit generality as long as the data is scaled) and its **scale** or **bandwidth**. To obtain the kernel density estimate, one can now smooth over (**??**) the set of delta functions that represent the observations or, equivalently, sum over normalized kernels that are centered at the observations (Fig. 4.4):

$$\begin{aligned} \hat{p}(x) &= \frac{1}{\mathfrak{n}} \int k(x-h;h) \sum_{i=1}^{\mathfrak{n}} \delta(h - x_i)\, dh \\ &= \frac{1}{\mathfrak{n}} \sum_{i=1}^{\mathfrak{n}} k(x - x_i; h) \quad \text{with} \quad \int k(x;h)\, dx = 1 \end{aligned}$$

Note that, in contrast to eq. (4.2), we have now normalized by the number of samples $\mathfrak{n}$: histograms usually give absolute counts, whereas density estimates must integrate to one, according to the axioms of probability theory. We have also made explicit the parameter h by writing $k(\cdot; h)$.

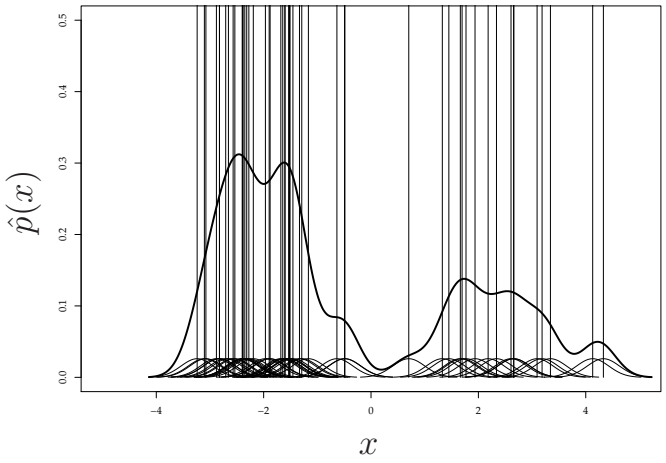


Figure 4.4: Kernel density estimation example: the location of the one-dimensional samples are sketched by vertical bars. The kernel density estimate (bold line) can either be seen as a smoothing of the sum of these delta distributions, or as sum of kernels that are centered at at these observations (shown at the bottom).

Being a simple linear superposition, the density estimate $\hat{p}$ inherits all properties from its summands: in particular, if the kernel is differentiable everywhere, then so is the density estimate; if the kernel is nonnegative, then so is the density estimate – meeting another requirement from said axioms.

Intuitively, we understand that proper choice of h is of crucial importance: making it too small will result in a density estimate that looks like a "stack of needles", or isolated peaks; and density estimates obtained from different samples from the *same* underlying distribution will look very different. On the other hand, making h extremely large will ultimately result in a estimate that reflects the shape of the employed kernel (typically, a single bump) more than that of the underlying data. In terms of the bias-variance tradeoff (see **??**), we can state that small h leads to large variance, whereas large h yields bias.

Statistics of kernel density estimation*

To make these considerations more quantitative, consider independent and identically distributed random variables $\mathbf{x}_{1:\mathfrak{n}} := \{\mathbf{x}_i | i = 1, \ldots, \mathfrak{n}\}$ with true, unknown probability density $p(x)$ which we would like to estimate from a finite sample comprising $\mathfrak{n}$ realizations $x_{1:\mathfrak{n}} := \{x_i | i = 1, \ldots, \mathfrak{n}\}$. The expected density estimate (as we repeatedly draw each of the $\mathfrak{n}$ variables from many

training sets) at position x becomes

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}_{1:\mathfrak{n}}}\hat{p}(x) &= \mathbb{E}_{\mathbf{x}_1}\dots\mathbb{E}_{\mathbf{x}_\mathfrak{n}}\hat{p}(x) = \mathbb{E}_{\mathbf{x}_{1:\mathfrak{n}}}\left(\frac{1}{\mathfrak{n}}\sum_{i=1}^{\mathfrak{n}}k(x-x_i;h)\right) && (4.2)\\
&= \frac{1}{\mathfrak{n}}\sum_{i=1}^{\mathfrak{n}}\mathbb{E}_{\mathbf{x}_i}k(x-x_i;h) && (4.3)\\
&= \frac{\mathfrak{n}}{\mathfrak{n}}\mathbb{E}_{\mathbf{x}_1}k(x-x_1;h) && (4.4)
\end{aligned}$$

The last equality stems from the fact that all random variables have the *same* distribution; we can hence replace the sum of expectations over these random variables with the $\mathfrak{n}$-fold multiple of a single one of these expectations, e.g. the first one. Inserting the definition of an expectation (appendix **??**), we finally find

$$\mathbb{E}_{\mathbf{x}_{1:\mathfrak{n}}}\left(\hat{p}(x)\right) = \int k(x-x_1;h)p(x_1)\,dx_1 \tag{4.5}$$

This is interesting indeed! The expected density estimate is simply the convolution of the true (but unknown) density with the kernel function, and the bias is hence the difference between the true and the smoothed probability density. The promise of this equation is that if we were to choose a Dirac function as a kernel, the expected density estimate would become identical to the true density estimate, and the bias would become zero! Of course, there would be a heavy toll to pay in terms of increased variance, as demonstrated next:

$$\mathrm{Var}_{\mathbf{x}_{1:\mathfrak{n}}}(\hat{p}(x)) := \frac{1}{\mathfrak{n}}\mathbb{E}_{\mathbf{x}_{1:\mathfrak{n}}}\left(\hat{p}(x)^2\right) - \frac{1}{\mathfrak{n}}\left(\mathbb{E}_{\mathbf{x}_{1:\mathfrak{n}}}\left(\hat{p}(x)\right)\right)^2 \tag{4.6}$$

We already know the last factor which is the square of eq. (4.5). Regarding the other term, we insert the definitions of expectation and kDE, reshuffle as before and finally obtain

$$\begin{aligned}
\mathrm{Var}_{\mathbf{x}_{1:\mathfrak{n}}}(\hat{p}(x)) &= \frac{1}{\mathfrak{n}}\int k(x-x_1;h)^2p(x_1)\,dx_1\\
&\quad -\frac{1}{\mathfrak{n}}\left(\int k(x-x_1;h)p(x_1)\,dx_1\right)^2 && (4.7)
\end{aligned}$$

Unsurprisingly, the variance of the estimator shrinks as the number of available samples $\mathfrak{n}$ grows. But how about the kernel width? A quantitative argument can be made [22] by Taylor expanding the true density. Simplifying this argument further to get an intuition, let us assume that we use the uniform kernel (in one dimension also called the box kernel: constant within a sphere, and zero outside) and that the true density is, to a good approximation, constant across the diameter of this primitive kernel. The integral can

then be restricted to the support of the kernel, and both kernel and density, assumed to be constant, can be pulled out of the integral. Remembering that kernels are normalized, we find that the variance becomes very large if $k(x;h) \gg p(x)$, which certainly holds for all "normal" densities as h shrinks and the kernel becomes ever more similar to a delta distribution.

Summing up, to minimize bias we should use a kernel that is as narrow as possible; but this will inflate the variance; and vice versa. Once again, we have encountered the **bias-variance tradeoff** that was already mentioned in passing in section 2.3. In kDE, the squared bias and variance add up to the mean squared error, defined as

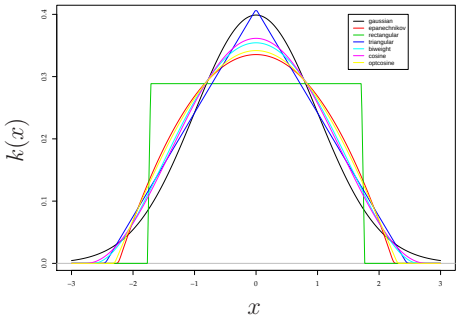
$$\begin{aligned} MSE(\hat{p}(x)) &= \mathbb{E}\left((\hat{p}(x) - p(x))^2\right) && (4.8)\\ &= \mathbb{E}\left((\hat{p}(x) - \mathbb{E}\left(\hat{p}(x)\right) + \mathbb{E}\left(\hat{p}(x)\right) - p(x))^2\right) && (4.9)\\ &= \mathbb{E}\left((\hat{p}(x) - \mathbb{E}\left(\hat{p}(x)\right))^2\right) \quad \text{variance} && (4.10)\\ &\quad + \left(\mathbb{E}\left(\hat{p}(x)\right) - p(x)\right)^2 \quad \text{squared bias} && (4.11) \end{aligned}$$

Parameter choice

Understanding the limiting cases of extremely broad or narrow kernels is all very well, but still leaves us with the eminently concrete problem of choosing an appropriate width for a real data set. In summary, this is an unsolved problem, characteristic of the obstacles that unsupervised learning encounters in general. There are cross-validation type procedures which choose the bandwidth h such that the average likelihood of an observation, given a kDE estimate constructed from all other observations, becomes maximal. However, these procedures fail for heavy-tailed distributions. In such distributions, single observations can lie very far from all others and require very broad kernels to be "explained" by the others. Alternatively, one may use an adaptive, or location-dependent, bandwidth. Here, the bandwidth may e.g. be set to the distance of the k-nearest neighbor. A proper number k[1] may then be chosen so as to maximize cross-validated likelihood, as just described.

Besides the bandwidth, we need to choose a shape for the kernel. Theoretical arguments show that the **Epanechnikov** kernel, an inverted truncated parabola, has maximum efficiency. It is also cheap to compute and allows for partitioning schemes such as kD-trees [1] thanks to its finite support. On the downside, the cusp at the limit of its range means that the resulting density estimate is not differentiable everywhere. Other kernels that are frequently used in practice are b-splines, a truncated cosine or the Gaussian (which has a somewhat inferior efficiency). The only popular kernel that has markedly worse efficiency is the uniform kernel; also, due to the piecewise constant nature of the resulting density estimate, it cannot be used in conjunction with

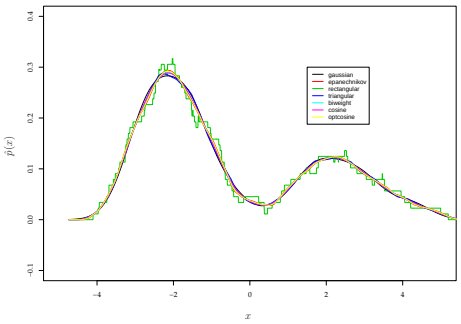
[1] To minimize confusion, we write k for the scalar number of nearest neighbors considered and $k(\cdot)$ for the kernel, a function of space.



(a)

Gaussian	95.12
Epanechnikov	100.00
Rectangular	92.95
Triangular	98.59
Biweight	99.39
Cosine	98.97
Optcosine	99.95

(b)



(c)

Figure 4.5: Different kernels **(a)** and their efficiencies **(b)**. Density estimates obtained from the samples in Fig. 4.4 using these kernels and an appropriate, user-defined bandwidth **(c)**.

gradient ascent schemes.

To guarantee that the resulting density estimate is always nonnegative, we need $k(x; h) \geq 0$. Kernels that violate this condition have been proposed to reduce bias, but have not found widespread use.

Altogether, kernel density estimates are more elegant than histograms in the sense that they eliminate the arbitrary dependence that histograms have on the shift of the first bin. Even so, they do not alleviate the problems associated with the curse of dimensionality: reliably estimating a density with high intrinsic dimensionality requires huge amounts of data. This is one of the motivations for feature selection or dimension reduction methods that seek to somehow extract the most relevant subspace. The prototypical method, principal components analysis, will be introduced soon, in chapter 4.4. Before, though, we will look at the relation of density estimation to cluster analysis.

4.3 Cluster analysis

To the human, Figs. 4.1, 4.4 suggest that there are perhaps two distinct mechanisms that have given rise to the observations. Such a search for patterns is so central to human cognition that we have no means of stopping it (with the possible exception of master meditators). The mechanism is so strong that humans hallucinate patterns, groups, or clusters in a set of points even if they are merely randomly distributed in space **??**. This points to the importance of cluster analysis for animate cognition. **Cluster analysis** seeks to find groups of observations that seem to "somehow belong together". Alas, the vagueness of this statement reflects the ill-posedness of the problem: while specific methods can be analyzed in detail, there are simply too many arbitrary decisions required to permit the development of a closed mathematical theory of all of cluster analysis. In fact, cluster analysis is in a sorry state compared to the supervised learning methods that we surveyed with justified confidence in chapters 2, 3. This should not come as a surprise. Assume you are given the task of "clustering the universe", with no additional information regarding what scale is of interest. Elementary particles, atoms, planets, solar systems and galaxies are all good candidates that share the property that they are relatively dense while being separated from each other by much empty space. As a corollary, you should be weary of publications that pretend to find a single "right" length scale, or number of clusters, by themselves with no further information. In fact, *all* clustering methods implicitly or explicitly need just that, a specification of the length scale of interest or of the desired number of clusters, even though some methods hide that need. It is of course possible to obtain a "scale-free" method by reiterating the analysis at *all* scales. This still leaves to the user the decision at which scale to make the cut.

Clusterings can either be defined by a partitioning of all observations into groups, the eponymous **clusters**, or by representative members, the **cluster centers** that may or may not coincide with one of the original observations. Some clustering techniques are **crisp**, i.e. each observation must belong to exactly one cluster. Others are **fuzzy**, i.e. one observation may have fractional memberships to several clusters.

One use of cluster analysis is in **exploratory data analysis**: given a big bunch of data, we would like to know "what's in it". Besides visualization, finding natural groups of observations the meaning or relevance of which can be established using domain knowledge is one important step. Especially when cluster centers are used to summarize an overwhelming amount of data, cluster analysis can also be seen as a mode of informed subsampling, or data compression: more on that in section 4.4. Again in an exploratory spirit, **hierarchical clustering methods** can help organize a wealth of information. Biological systematics are an important example, see Fig. 4.6 for an early example. Hierarchical methods are mostly crisp and always have

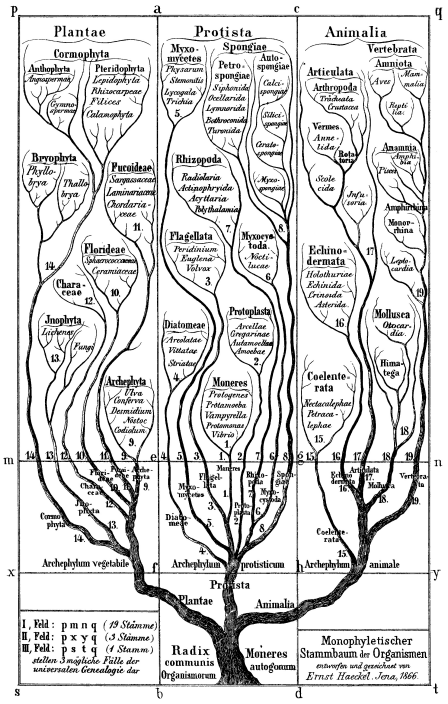


Figure 4.6: Tree of life as proposed by 19th century egotistic but influential German zoologist Ernst Haeckel [12]. Current research presumes a web, i.e. a graph *with* loops (see **??**), rather than a tree, of ancestry [1].

a scale parameter. Their defining characteristic is the guarantee that two samples that belong to one cluster at a fine scale will always be in the same cluster at coarser scales.

We will consider only two non-hierarchical crisp cluster analyses at this point, one based on partitionings and one on cluster representatives. For an important fuzzy technique, see **??**.

4.3.1 Mean shift clustering

We start with a "partitioning" method that groups observations by membership to the same local maximum of a kernel density estimate. To this end, we could start an ascent along the gradient of a kernel density estimate:

$$\nabla_x \hat{p}(x) \quad = \quad \sum_{i=1}^{n} \nabla_x k(x - x_i) \tag{4.12}$$

and then group, after convergence, all points that have reached the same point in feature space, i.e. the same density maximum.

It turns out that the same result can be achieved by the **mean shift algorithm**. Since the relation is interesting, we will discuss it in some detail.

While mean shift was first proposed in 1975 [8], it took 20 years until Yizong Cheng clarified the precise nature of the connection to density estimation [5].

Given the observations $x_{1:\mathfrak{n}}$, define the local mean $lm\left(x^{(t)}\right)$ around $x^{(t)}$ as

$$lm(x^{(t)}) \quad = \quad \frac{\sum_i msk\left(x^{(t)} - x_i\right) x_i}{\sum_i msk\left(x^{(t)} - x_i\right)} \tag{4.13}$$

where msk stands for "mean shift kernel". A query point position $x^{(t)}$ can now be updated iteratively according to $x^{(t+1)} = lm(x^{(t)})$. The difference vector $lm\left(x^{(t)}\right) - x^{(t)}$ is called the "mean shift". In the original implementation, all observations $x_{1:\mathfrak{n}}$ were updated after each iteration [8]. This is called a "blurring process". In the version discussed here, however, the $x_{1:\mathfrak{n}}$ remain fixed.

If a total of $\mathfrak{n}$ queries are initialized at the sample positions $x_{1:\mathfrak{n}}$ and shifted iteratively according to (4.13) until convergence, then a clustering for all $\mathfrak{n}$ observations results.

If we want the stationary points of the mean shift iterations to coincide with the local maxima of the kernel density estimate, we need the gradient of the latter and the mean shift vectors to be parallel, and hence identical up to some multiplicative constant c:

$$\begin{aligned} c\nabla_x \hat{p}(x) &= lm(x) - x \\ c\sum_i \nabla_x k(x - x_i) &= \frac{\sum_i msk(x - x_i) x_i}{\sum_i msk(x - x_i)} - x \\ &= \frac{\sum_i msk(x - x_i)\left(x_i - x\right)}{\sum_i msk(x - x_i)} \end{aligned} \tag{4.14}$$

We restrict ourselves to isotropic kernels so that $k(x) = k(||x||)$ and furthermore reparametrize the kernel $k(x)$, in a way that will prove beneficial, in terms of the "kernel profile" $kp(u) := k(x)$ with $u := x^2$. This allows $\nabla_x k(x)$ to be rewritten as $\nabla_u kp(u)\frac{du}{dx} = \nabla_u kp(u) 2x$ so that

$$\nabla_x \hat{p}(x) = \sum_i \nabla_x k(x - x_i) = -2\sum_i \nabla_{u_i} kp(u_i)(x_i - x) \text{ with } u_i = ||x - x_i||^2 \tag{4.15}$$

Inserting this into eq. (4.14) gives

$$-2c\sum_i \nabla_{u_i} kp(u_i)(x_i - x) \quad = \quad \frac{\sum_i msk(x - x_i)\left(x_i - x\right)}{\sum_i msk(x - x_i)} \tag{4.16}$$

$$c(x)\sum_i \nabla_{u_i} kp(u_i)(x_i - x) \quad = \quad \sum_i msk(x - x_i)\left(x_i - x\right) \tag{4.17}$$

where the denominator on the right hand side of the first line and the factor of -2 have been absorbed into a multiplicative constant $c(x)$ that now has become a function of space.

shadow kernel $k(\cdot)$	mean shift kernel $msk(\cdot)$
Epanechnikov	Box kernel
Biweight	Epanechnikov
Cosine	Sinc (central lobe of)
Gaussian	Gaussian

Table 4.1: Performing iterated mean shifts with kernel $msk(\cdot)$ amounts to a gradient ascent on a kernel density estimate with shadow kernel $k(\cdot)$.

The last equation finally reveals our principal result:

$$\nabla_u kp(u) \propto msk(x) \tag{4.18}$$

In words, the mean shift kernel $msk(x)$ that should be constructed to obtain the identical clustering as given by a gradient ascent on a density estimate with the "shadow kernel" (Cheng's terminology) $k(\cdot)$ is obtained by differentiating the kernel profile. Some specific examples are given in Table 4.1. As in other respects (**??**), the Gaussian kernel is unique in that it is the only kernel who is its own "shadow" [5].

4.3.2 k-means clustering

Clusterings defined in terms of a partitioning are costly: most procedures require a full $\mathfrak{n} \times \mathfrak{n}$ proximity or dissimilarity matrix. Clusterings defined in terms of k **representatives** or **cluster centers**, on the other hand, require only $k \times \mathfrak{n}$ proximity or membership matrices, with $k \ll \mathfrak{n}$. The prototypical method is **k-means clustering**, which works as follows: the k cluster centers (hence the name) are initialized randomly, e.g. at the location of k randomly selected observations. Then, the following steps are iterated:

- Compute the binary indicator matrix $M \in \{0,1\}^{k \times \mathfrak{n}}$, $\sum_i [M]_{i,j} = 1$ that states which cluster (center) each observation is assigned to, based on smallest Euclidean distance. Possible ties can be broken randomly.
- Move each cluster representative to the center of mass of "its" observations.

It can be shown [4] that k-means converges almost surely to a local minimum, and that it amounts to a Newton optimization of the squared error in eq. (4.19). The improvement of the clustering over time can be tracked by the evolution of that same squared error.

If all observations are stored in a matrix $X \in \mathbb{R}^{\mathfrak{p} \times \mathfrak{n}}$ and the cluster centers are summarized in a matrix $C \in \mathbb{R}^{\mathfrak{p} \times k}$, then the average of all squared

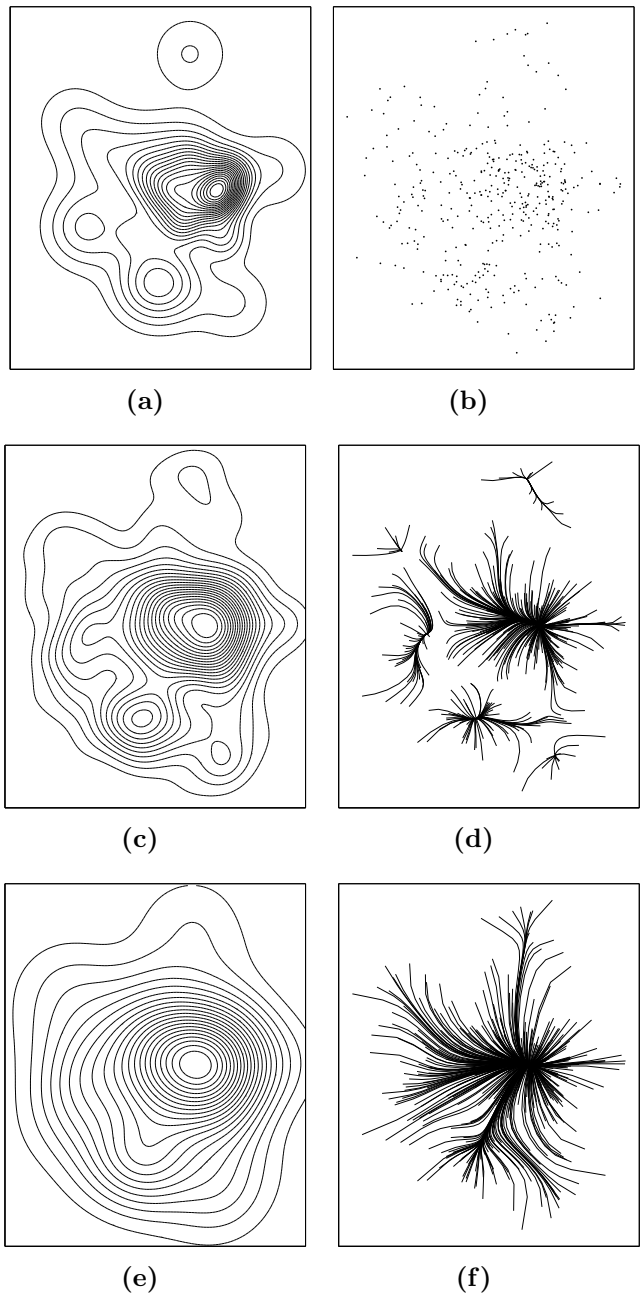


Figure 4.7: **(a)** True density isocontours and **(b)** sample of 400 points drawn from true density. **(c)** Kernel density estimate based on above sample with Gaussian kernel and **(d)** corresponding mean shift trajectories. **(e)**-**(f)**As above, with larger bandwidth. The mean shift iterations result in six clusters and a single one, respectively.

Euclidean distances from each observation to "its" cluster center is given by

$$\mathfrak{E} = \frac{1}{\mathfrak{n}}||CM - X||_F^2 \tag{4.19}$$

Actually, the minimization of this error has been a driving force in the development of the k-means, or Linde-Buzo-Gray or Lloyd algorithm, as it is better known in communications: in a lossy compression scheme, it is a natural measure for the loss of information – more on that in the next section.

This simple algorithms owes its popularity to its ease of implementation and decent results; but there are pitfalls. Specifically, the algorithm typically converges to a local minimum (especially in high dimensions) so that only the best solution out of multiple restarts should be used. Due to the non-robust error function (4.19), it is very sensitive to outliers that may recruit one precious cluster center to represent a single observation. Finally, the cluster centers typically do not coincide with one of the original observations. This is an obstacle in some applications, but can sometimes be worked around by replacing the cluster centers with their closest neighbors among the original data. Note, however, that this solution is not optimal. Algorithms that seek to minimize eq. (4.19) with respect to C in the reduced solution space that is limited to the original observations are known as **k-medoid** clustering []. Further variants such as the "Isodata algorithm" use heuristic splitting and merging schemes to find better local minima than plain k-means.

An example application – clustering of the colors in an image – is shown in Fig. 4.8. Incidentally, it also shows how remote naïve machine vision is from human perception: to a human, tiger and bait clearly stand out as distinct objects, in front of a background that is so bland it is not even worth mentioning. However, if we use four color clusters, then two are used to explain the background (and part of the predator), with another two describing texture. It takes many more clusters to separate tiger and prey. Imagine how much more difficult the situation is when the background is cluttered, or when different objects do not have distinct colors... And yet there is hope, which will be nourished in chapter **??**.

4.3.3 Vector quantization (VQ)

Consider the example from Fig. 4.8, and assume that we want to somehow coarsen the color representation to reduce storage requirements. The original image uses a triple $\{0, \ldots, 255\}^3$ of red, green, blue values (i.e. 3 bytes, 24 bits) to characterize each pixel. A no-brainer would be to reduce the color resolution, e.g. to $\{0, \ldots, 15\}^3$ (i.e. 12 bits). However, Fig. 4.8 **(b)** hints to more alluring alternatives. Indeed, the image at hand (as, in fact, most images) does not sample color space evenly; on the contrary, only a small part of the space is occupied. With a regular grid of possible color values, we would hence be "wasting" most of the grid points, and hence bits, while

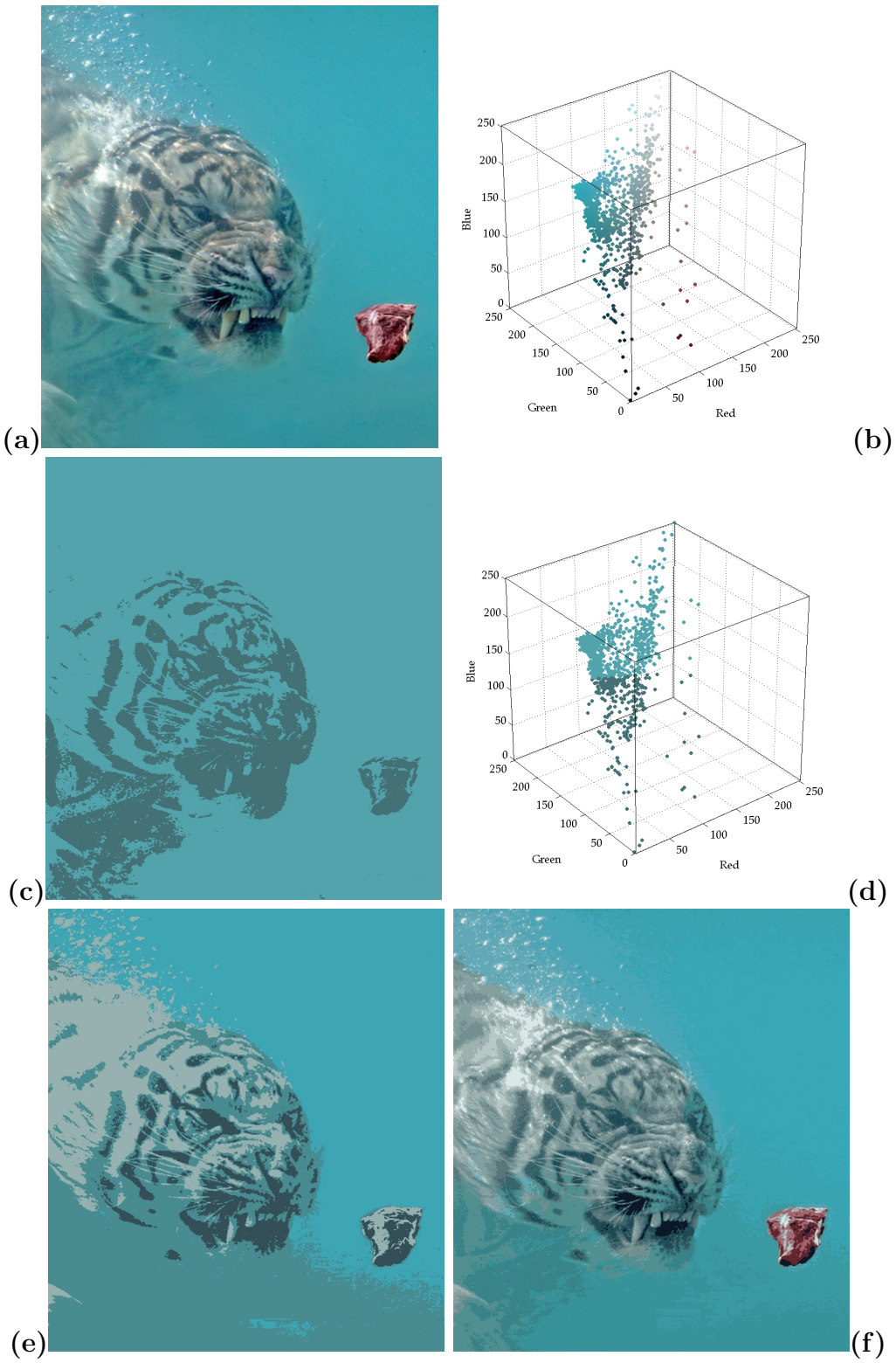


Figure 4.8: **(a)** Original image by an unknown artist. Even though there are perceptually few colors and these are poorly saturated, the image comprises more than 53'000 distinct colors. **(b)** Representation of image in RGB color space: each pixel is represented by a dot. To avoid occlusion, the image has been subsampled. **(c)** Color-compressed version of original image with k=2 colors. **(d)** Cluster membership is indicated by the color of each dot. **(e)**, **(f)** Color quantized images with k=4, 20.

crowding in on a few select others. Intuitively, it seems we are better advised to spend our limited budget of grid points, or prototypes, where the action is. How to best do this is precisely the subject of vector quantization.

In **vector quantization (VQ)**, the task is to find a set of prototypes, or **code vectors**, that approximate the underlying distribution well[2]. Specifically, the goal is to minimize the average squared error or loss (eq. (4.19)) incurred when approximating each observation with its closest code vector. This goal is typically approximated by using the k-means algorithm which finds a local minimum of this compression error. When considering the savings from this lossy compression, we need to remember that there is an overhead: the set of all code vectors – the **code book** (or, in the example of Fig. 4.8: the **color palette**) – first needs to be determined for a given source, and then stored or transmitted; but just once! Overall, this strategy usually makes for huge savings and permeates all of today's telecommunication.

From a different perspective, the code book found in vector quantization is one way of summarizing a (potentially high-dimensional) distribution in a succinct way, by storing only the code book C and the row sums of M (telling us how many examples are represented by each code vector). The ability to compactly summarize a high-dimensional distribution makes VQ a building block of the utmost importance in many modern vision system. For example, in object recognition systems, one may detect (**??**) and characterize (**??**) a set of interest points. The resulting descriptors span a high-dimensional feature space and in **bag of words** systems (**??**), discriminating objects amounts to learning the difference between high-dimensional distributions. As stated before, estimating a high-dimensional distribution precisely is a formidable problem; and anyway given a limited training set of objects we cannot afford to use very high-dimensional representations for objects, or we are likely to overfit the data (**??**). It is here that vector quantization comes in handy.

As a second example, consider texture recognition (**??**). A typical approach is to partition a given image into patches, and represent each patch by a set of filter outputs [19] or simply in terms of its pixel intensities [26]. After vector quantization, a complete image can be summarized in terms of a single vector that states how many patches are represented by each code vector, see page **??**. These relatively low-dimensional representations of fixed dimensionality can then be fed into any of the classifiers that we have already see or will discuss later on.

Finally, note the close relation to histograms: a vector as just described *is* a histogram, the only difference being that now the bins are no longer uniform but have been adapted to the data.

[2]The term "vector quantization" is derived from "quantization", the one-dimensional special case that has historical precedent.

4.4 Principal components analysis (PCA)

In vector quantization, given a set of $\mathfrak{p}$-dimensional observations $x_{1:\mathfrak{n}}$ that are summarized in $X \in \mathbb{R}^{\mathfrak{p}\times\mathfrak{n}}$, the problem is to

$$\min_{C,M} ||CM - X||_F^2 \quad \text{s.t.} \qquad C \in \mathbb{R}^{\mathfrak{p}\times k} \tag{4.20}$$

$$M \in \{0,1\}^{k\times\mathfrak{n}} \tag{4.21}$$

$$1^T M = 1 \tag{4.22}$$

a local minimum of which is found by k-means clustering. The restriction to integer memberships is what makes the solution sparse – each observation is approximated in terms of a single code vector – but also what makes finding the global optimum hard. We could consider a relaxed problem formulation in which each observation is explained as linear combination of k (or, as we write in this section: $\mathfrak{r}$[3]) representatives, without the integrality constraint:

$$\min_{U,W} ||UW^T - X||_F^2 \quad \text{s.t.} \qquad U \in \mathbb{R}^{\mathfrak{p}\times\mathfrak{r}} \tag{4.23}$$

$$W^T \in \mathbb{R}^{\mathfrak{r}\times\mathfrak{n}} \tag{4.24}$$

This is just what principal components analysis (PCA) does. The result is a **bilinear decomposition** $X \approx UW^T$, so called because it decomposes X into two factors. The columns of U now are the representative vectors, here called **principal components (PCs)** or **loadings**, and the columns of W^T tell us how much of each representative vector we need to approximate a given observation; these coefficients are called **scores**.

To gain more insight into the meaning of this optimization problem, let us carry this approximation to the extreme and work with a single principal component only. That is, we request that $\mathfrak{r} = 1$ and write

$$X \approx u_1 w_1^T \tag{4.25}$$

where $u_1 := (U_{\cdot,1})$ and $w_1 := W_{\cdot,1}$. By construction, this approximation has rank $\mathfrak{r} = 1$.

Expanding the Frobenius norm, we find

$$\begin{aligned} ||X - u_1 w_1^T||_F^2 &= \operatorname{tr}\left(\left(X - u_1 w_1^T\right)^T \left(X - u_1 w_1^T\right)\right) \\ &= \operatorname{tr}(X^T X) - 2\operatorname{tr}\left(w_1 u_1^T X\right) + \operatorname{tr}\left(w_1 u_1^T u_1 w_1^T\right) \end{aligned} \tag{4.26}$$

The first term on the right hand side is independent of u_1, w_1, so we omit it in the following. Exploiting the invariance of the trace operator under cyclic

[3] With respect to eq. (4.22), we replace the symbol k with $\mathfrak{r}$ (for "rank"), and we also replace C by U and M by W^T to obtain a notation that is reminiscent of the SVD convention introduced in section 4.4.1

permutation ($\mathrm{tr}(CD) = \mathrm{tr}(DC)$ if C, D^T have the same dimensions), we can write

$$-2\mathrm{tr}\left(u_1^T X w_1\right) + \mathrm{tr}\left(u_1^T u_1 w_1^T w_1\right) \tag{4.27}$$

The arguments of the trace operators have now become scalars, so this is simply

$$-2u_1^T X w_1 + u_1^T u_1 w_1^T w_1 \tag{4.28}$$

Since we want to minimize this expression with respect to u_1, w_1, its derivative at the optimum should be zero:

$$\frac{d}{du_1}\left(-2u_1^T X w_1 + u_1^T u_1 w_1^T w_1\right) = -2Xw_1 + 2u_1 w_1^T w_1 \stackrel{!}{=} 0 \tag{4.29}$$

$$u_1 = \frac{Xw_1}{w_1^T w_1} \tag{4.30}$$

$$\frac{d}{dw_1}\left(-2u_1^T X w_1 + u_1^T u_1 w_1^T w_1\right) = -2X^T u_1 + 2u_1^T u_1 w_1 \stackrel{!}{=} 0 \tag{4.31}$$

$$w_1 = \frac{X^T u_1}{u_1^T u_1} \tag{4.32}$$

Inserting the latter into eq. (4.28) leaves only

$$-\frac{u_1^T X X^T u_1}{u_1^T u_1} \tag{4.33}$$

to be minimized. Flipping the sign turns this into a maximization problem, so that our goal now is to maximize

$$\lambda_1 := \frac{u_1^T X X^T u_1}{u_1^T u_1} \tag{4.34}$$

a form which is also known as **Rayleigh quotient**. But

$$XX^T u_1 = \lambda_1 u_1 \tag{4.35}$$

is the classical form of an eigenproblem, which admits only the eigenvectors and eigenvalues as possible solutions for u_1 and λ_1, respectively. Among all these admissible solutions, we are interested in the largest one, which is given by the largest eigenvalue of XX^T, and its corresponding eigenvector. This result lies at the heart of principal components analysis. The finding that the largest / smallest value that can be attained by a Rayleigh quotient is given by the largest / smallest eigenvalue is the subject of the **Rayleigh-Ritz theorem** [1].

Since we can always multiply an eigenvalue with some constant $c \neq 0$ and at the same time divide the corresponding eigenvector by the same c without

violating the eigen equation, we can without loss of generality normalize the eigenvector such that $u_1^T u_1 = 1$ and absorb into λ_1 the multiplicative constant that guarantees satisfaction of eq. (4.35).

With this choice of normalization, we can summarize: the first principal component, i.e. the best rank 1 approximation to X as measured by eq. (4.26) is given by the eigenvector u_1 pertaining to the largest eigenvalue of XX^T. In this reduced representation, *the original observations are approximated* by the scores

$$w_1 = X^T u_1, \quad u_1^T u_1 = 1 \tag{4.36}$$

that is, *by an orthogonal projection onto the normalized first principal component.*

There are more ways of motivating this important result: the first principal component also specifies that one-dimensional subspace that leaves the smallest squared residuals when projecting the original observations onto it. This observation follows from the fact that we can rewrite the original error function as

$$\mathfrak{E}^2 = ||X - u_1 w_1^T||_F^2 = ||X - u_1 u_1^T X||_F^2 = || \left(I^{\mathfrak{r}} - u_1 u_1^T\right) X||_F^2 \tag{4.37}$$

where $u_1 u_1^T$ is the projection operator that casts any datum onto u_1 and $u_1 u_1^T X$ is the orthogonal projection of all observations X onto the subspace u_1. The argument of the norm hence holds all residual or difference vectors, the difference between the original data and their projections; and the norm itself gives the sum of the squared lengths of all these difference vectors.

Another interpretation of the result is that it gives that single direction which has the maximal spread of the observations along it. Since the principal component has been normalized, the score directly measures the distance from the origin. The sum of the squared scores, which we need to maximize if we want to obtain the largest possible spread, is

$$w_1^T w_1 = u_1^T X X^T u_1 \tag{4.38}$$

which is just the expression we already maximized in eq. (4.34).

Now, if the best rank 1 approximation to X is given by the first eigenvector of XX^T, what is the best rank $\mathfrak{r}$ approximation? You most probably have a guess, and it most probably is correct. But to be on the safe side, and since we will rely heavily on the soundness of the result: let us do the maths.

4.4.1 Interlude: singular value decomposition (SVD)

We will make use of the **singular value decomposition (SVD)**, a matrix decomposition

$$X = USV^T \tag{4.39}$$

of great importance. It can be shown to be unique and always exist [9]. $U \in \mathbb{R}^{\mathfrak{p}\times\mathfrak{r}^*}$ and $V \in \mathbb{R}^{\mathfrak{n}\times\mathfrak{r}^*}$ with $\mathfrak{r}^* = \text{rank}(X)$ are orthogonal (i.e., $U^TU = I, V^TV = I$) and their columns hold the **left singular vectors** and the **right singular vectors**, respectively. The $\mathfrak{r}^* \times \mathfrak{r}^*$ matrix S is diagonal and holds the **singular values** $\sigma_{1:\mathfrak{r}^*}$, sorted in decreasing order. The singular values are given by the square roots of the eigenvalues of either[4] XX^T or X^TX, while the left and right singular vectors are given by the respective eigenvectors, as seen by inserting the definition of the SVD:

$$XX^T \overset{(4.39)}{=} USV^TVSU^T = US^2U^T \tag{4.40}$$

$$XX^TU = US^2 \tag{4.41}$$

$$X^TX = VSU^TUSV^T = VS^2V^T \tag{4.42}$$

$$X^TXV = VS^2 \tag{4.43}$$

The fact that U, V hold the eigenvectors of symmetric matrices also vindicate the above claim of their orthogonality.

The truncated SVD

$$US_{\mathfrak{r}}V^T \tag{4.44}$$

in which U, S, V are derived from X except that the last $\mathfrak{r}^* - \mathfrak{r}$ singular values in S have been set to zero, gives the best rank $\mathfrak{r}$ approximation to X [9].

To verify, solve

$$\min_{X_{\mathfrak{r}}} ||X - X_{\mathfrak{r}}||_F^2 \tag{4.45}$$

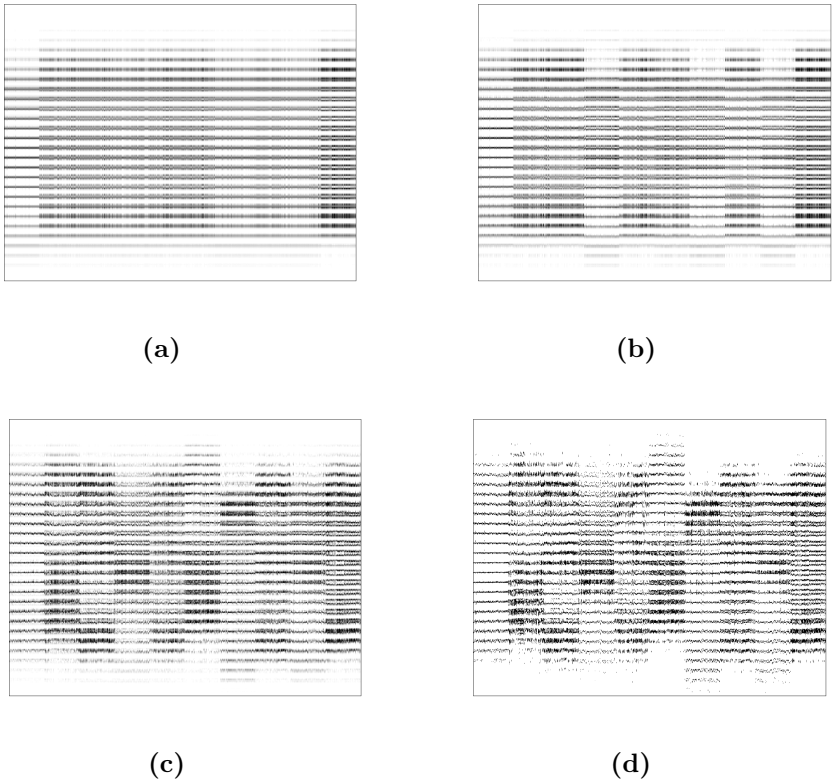
We decompose both the known X and unknown X_r in terms of an SVD: $X = USV^T$ and $X_{\mathfrak{r}} = ABC^T$. The Frobenius norm is invariant under multiplication with an orthogonal matrix (because $||A||_F^2 = \text{tr}(AA^T) = \text{tr}(AUU^TA^T) = ||AU||_F^2$), so that

$$\mathfrak{E} = ||X - X_{\mathfrak{r}}||_F^2 = ||USV^T - ABC^T||_F^2 = ||S - U^TABC^TV||_F^2 \tag{4.46}$$

B and S are, by the properties of the SVD, diagonal matrices. If we choose $A = U$ and $C = V$, the diagonal entries of B cannot "stain" the off-diagonals of the product U^TABC^TV. Since such off-diagonal elements would necessarily increase $\mathfrak{E}$ without any positive side-effects, we may as well set $A = U$ and $C = V$ in the following. This leaves, as only degree of freedom, the choice of the singular values in B. Only $\mathfrak{r}$ out of these may be non-zero if we wish to restrict the rank of $X_{\mathfrak{r}}$ to $\mathfrak{r}$. With our choice of left and right singular vectors, the approximation error $\mathfrak{E}$ is given by $\sum_{i=1}^{\mathfrak{r}^*} ([S]_{i,i} - [B]_{i,i})^2$. The best choice is $S_{\mathfrak{r}} = B$ with

$$[B]_{i,i} = \begin{cases} \sigma_i & 0 \le i \le \mathfrak{r} \\ 0 & \mathfrak{r} < i \le \mathfrak{r}^* \end{cases} \tag{4.47}$$

[4] Note that XX^T and X^TX have the same eigenvalues, as shown by the following equation.



(a) **(b)**

(c) **(d)**

Figure 4.9: SVD approximations of rank **(a)** 1, **(b)** 2, **(c)** 10 to a matrix of original observations **(d)**. The columns of the latter are vectorized patches of 100 exemplars each of the digits $0, \ldots, 9$.

Reiterating, the best rank $\mathfrak{r}$ approximation to X is given by the SVD of X, where only the $\mathfrak{r}$ largest singular vectors are retained and all others set to zero.

4.4.2 Reprise: Principal component analysis

The results just obtained tell us more about PCA. The first principal component appears in a new light, as the best rank 1 approximation to the original data. By the new results, the best rank $\mathfrak{r}$ approximation to the data is given by the scores

$$W^T = SV^T = U^T X \tag{4.48}$$

in a basis of the eigenvectors pertaining to the $\mathfrak{r}$ largest eigenvalues of XX^T. It is truly justified to speak of a "basis" because said eigenvectors are orthogonal.

The multiplication of X with the orthonormal matrix U^T can be interpreted as a rotation (and perhaps reflection) into a new basis; and the elimination of the smallest $(\mathfrak{r}^* - \mathfrak{r})$ singular values can be seen as a projection to the first $\mathfrak{r}$ axes of this basis that span an optimal subspace: optimal in the sense of the best rank $\mathfrak{r}$ linear approximation to the original data. The basis is found in a **data-dependent** fashion by finding the SVD or solving an eigen problem. This is in contrast to data-independent basis functions as used in the cosine transform which forms the basis of JPEG compression (see **??**).

Several observations are in order: firstly, the sign of the eigenvectors is arbitrary and (except by chance) not preserved if the data is perturbed in any way, or if the PCA is performed with a different computing library or on a different computer. This means that care is required when comparing the scores of observations for which separate PCAs have been carried out. Secondly, the scores themselves can be positive *or* negative. This is in contrast to a k-means approximation, and also in contrast to bilinear decompositions with a positivity constraint (see **??**). Negative scores are not meaningful in some applications, and generally speaking the principal components, being linear combinations of the original axes, are often hard to interpret.

In the formulation used so far, the approximation plane spanned by the principal components (PCs) must necessarily go through the origin. This unnecessary limitation can be lifted by first subtracting the **mean vector**

$$\mu = \frac{1}{\mathfrak{n}} \sum_{i=1}^{\mathfrak{n}} x_i = \frac{1}{\mathfrak{n}} X 1^{\mathfrak{n} \times 1} \tag{4.49}$$

from all observations (in matrix notation: $X - \mu 1^{1 \times \mathfrak{n}}$) and then performing PCA. To obtain the approximation of the original data, the mean vector of course needs to be added back in, so that e.g. for the approximation of the ith observation,

$$\hat{x}_i = \mu + \left[U(SV)^T \right]_{\cdot,i} \tag{4.50}$$

For a concrete example, let us refer back to the digit recognition example from section 2.2. There, we worked with the simplest feature space conceivable: with image space, i.e. the grey value of each pixel was taken as a separate feature. As discussed in section 2.4.1, this space is convenient to define but has none of the desired invariances built-in. In addition, it is too large for the relatively narrow class of images that are considered in a specific application. In image space, an image of a digit is as good as that of a face or any other image. By admitting a feature space with an unnecessarily high dimensionality, we needlessly increase the amount of storage and of computations required; and we may even make learning more difficult.

To avoid these problems, it may help to cast the observations to a lower-dimensional subspace. As discussed above, PCA finds the best such space

according to a number of equivalent criteria. See Fig. 4.10 for a set of increasingly faithful approximations to the original high-dimensional observations. In this example, each sample is one gray value image, i.e. a 2D matrix; but in PCA, each sample should be represented by a single column of X. A simple way out is to **vectorize** each image patch, i.e. to rearrange[5] all the gray values into a single column vector, and to apply the reverse operation when showing the result.

A **scree plot** as in Fig. 4.10(b) is often used to decide on an appropriate number of components; in particular, one is interested in **shoulders**, i.e. sudden drops in the magnitudes of the eigenvalues that indicate that the following components may be of lesser importance. Regarding "importance", a word of warning is appropriate here: the new basis functions are ranked merely according to the spread of the data along those axes; these are often, but not necessarily, the most informative axes. For instance, Fig. 4.11 shows two class densities. If the aim is discrimination, the first PC is not informative at all. Such configurations can arise in the real world, for instance if the observations are subject to random multiplicative noise which "stretches" the observations along the bisectant. This is typical for spectral data as encountered in chemometrics and explains the importance of proper **normalization** of the observations. Unfortunately, what is "proper" depends on the specific type of noise or random distortion encountered in a concrete application, and domain expertise or experimentation may be required to correct for it.

Normalization also plays a crucial role if the individual features are of a different nature: for instance, if one feature is frequency and another is weight, it certainly matters if the former is measured in Hz or cm^{-1} and if the latter is given in kg or mg: these correspond to a nonlinear transformation of the frequency and rescaling of the weight axis, and do influence the direction of the resulting principal components. This is perhaps most intuitive through the mechanical analogy in which PCs correspond to the axes of the inertial tensor, in this case of the "data cloud". One way out is to normalize by dividing each feature by its standard deviation; but this can also entail loss of relevant information in some applications. In summary, adequate scaling of the axis is required before applying PCA.

PCA is so important that it has been reinvented in different fields, where it is known under names such as **Hotelling transform**, **discrete Karhunen-Loève transform**, **whitening** (in signal processing), proper orthogonal decomposition (fluid dynamics), **latent semantic indexing (LSI)** (document retrieval), and others. "Whitening" refers to the fact that in the new basis, the features are uncorrelated or "decorrelated" – this is sometimes used to turn a signal with correlated or "colored" noise into one with uncorrelated or "white" noise, e.g. in matched filtering []. "**Sphering**" means that the coefficients in the new basis are divided by the respective singular values to

[5] In matlab, the appropriate command is `reshape(patch, [], 1)` or `patch(:)`.

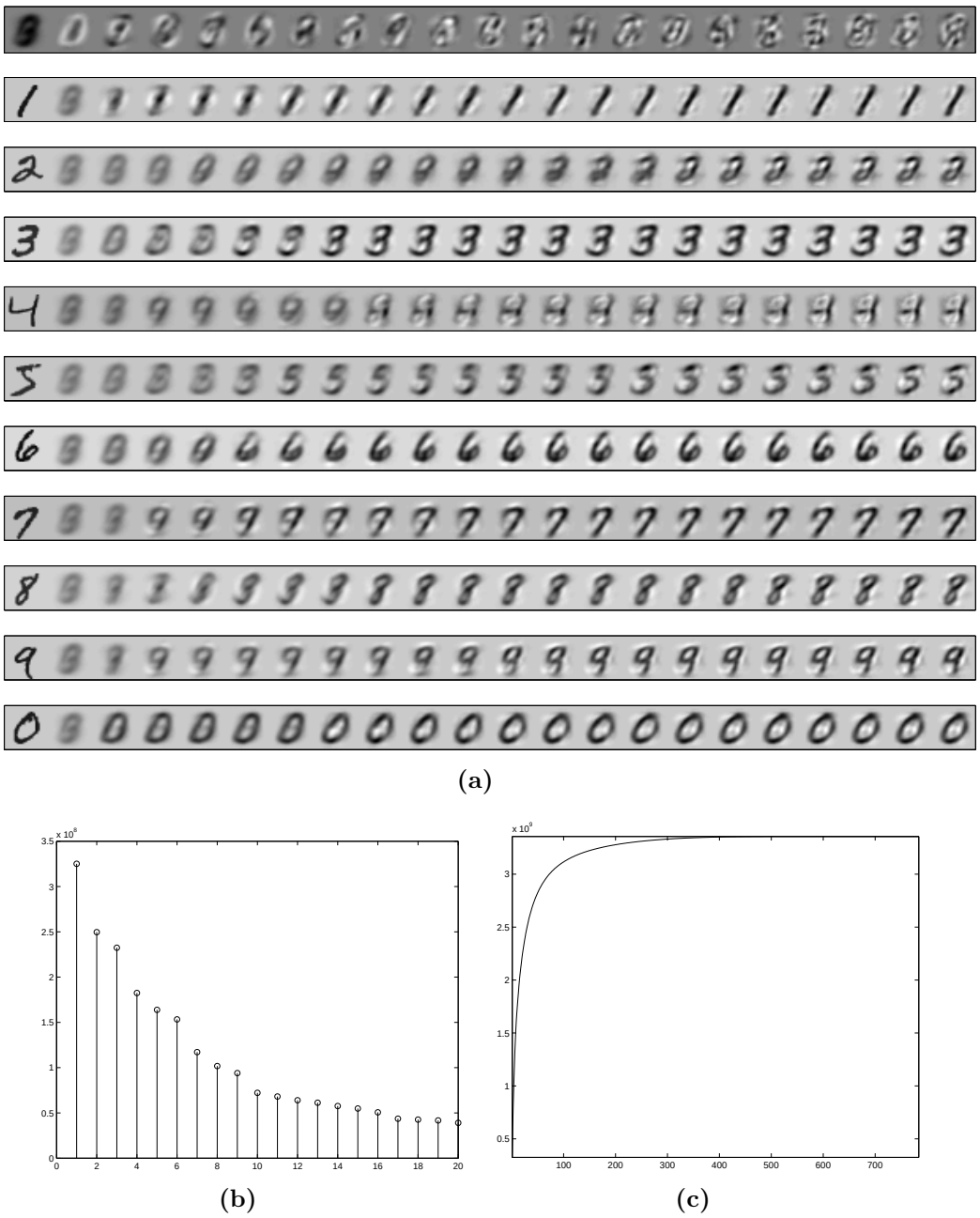


Figure 4.10: Principal component analysis of a data set comprising 100 MNIST samples of each digit. **(a)** The first row shows the mean image, followed by the 20 first principal components (note that the sign of the principal components is arbitrary). The following rows show an arbitrarily chosen training sample i as well as its reconstruction $U[W^T]_{:,i}$ using an increasing number of principal components. The digits in the last row are described by using the mean image plus 20 coefficients (as opposed to $28 \times 28 = 784$ gray values in the original patch). **(b)** The first 20 eigenvalues and **(c)** cumulative sum of all 784 eigenvalues. Around 400 eigenvectors are enough to span the full space of the 1000 training observations without loss; and a much smaller number is sufficient to obtain a good approximation.

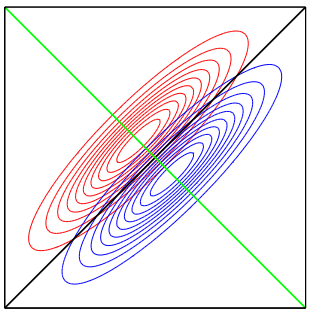


Figure 4.11: (Note to self: Replace this with an example of samples, rather than distributions). Toy example illustrating the first principal component (black line) and the linear discriminant (green line) for the two class densities in a binary classification problem. In this contrived example, the first PC is not discriminative at all, but all relevant information resides in the second PC.

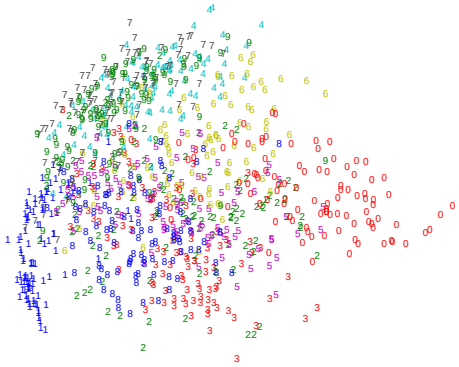


Figure 4.12: 2D scatter plot summarizing the original observations (image patches from the MNIST database that each showed one hand-drawn digit). Abscissa and ordinate are given by the two first scores of a PCA analysis. Even though PCA knows no labels, each point is marked by a symbol corresponding to its true class.

obtain, in the new basis, an uncorrelated data set with unit variance along all axes.

The truncation of the basis realizes a compression. In the last column of Fig. 4.10(a), a compression ratio of $1 - 20/784 \simeq 97.5\%$ is realized (in specifying this number, we have ignored the overhead for storing the mean pattern μ and the basis functions U). A compressed version of the original observations is shown as a scatter plot in Fig. 4.12. Each image patch has now been reduced to a single point in a 2D space.

Projection to principal components has proven effective in improving classification accuracy in a wide array of simple appearance-based problems, e.g.in face recognition under controlled pose and lighting. In the image processing

```
function [Xmean, LSV, RSV, eigenvalues] = iiapr_pca(X, trank);
% input
%    X     -- pxn matrix holding n observations with p dimensions each
%    trank -- target rank of approximation
%
% output
%    Xmean -- column-wise mean of X
%    LSV   -- "trank" first left singular vectors
%    RSV   -- "trank" first right singular vectors, transposed
%    eigenvalues -- *all* eigenalues

Xmean = mean(X, 2);

X = X - Xmean * ones(1, size(X, 2));
[LSV, eigenvalues, RSV] = svd(X, 'econ');
% assuming that singular values are already sorted in decreasing order
eigenvalues = diag(eigenvalues).^2;
RSV = RSV(:, 1:rank);
LSV = LSV(:, 1:rank);
```

Figure 4.13: PCA algorithm. The columns of the matrix `LSV` are the new basis vectors, or "loadings", given by the left singular vectors of an SVD decomposition. The columns of the matrix `RSV` are the approximate coefficients of the observations in the new basis, the "scores".

community, PCA has thus been rebranded as "eigenfaces" [25]. **Eigenfaces** are nothing but the loadings, corresponding to the first row of Fig. 4.10.

Overall, with PCA you have now met one of the most fundamental pattern recognition techniques there is.

4.5 Summary

- "Unsupervised learning" summarizes data analysis strategies that disregard labels, even if these are available. This does not rule out the selection of a subset of points based on their labels in a previous, supervised step.
- The density of points in feature space can be estimated using histograms or a kernel density estimate. In either case, an extremely large number of observations is required to obtain reliable estimates in high dimensions.
- In kernel density estimation, the expected density is the true density convolved with the kernel. Large kernels lead to bias, small kernels to

high variance.

- Cluster analysis finds natural groupings in data. The number of clusters or a scale of interest needs to be selected manually. Crisp schemes assign each observation to a single cluster, whereas fuzzy methods allow fractional cluster memberships.
- k-means is a crisp clustering that seeks to minimize the sum of squared residuals when approximating each observation with its closest cluster center. Only a local minimum of this target function is found.
- Principal components analysis (PCA) finds the best linear subspace that approximates the observations well. It can be seen as a dimension reduction or data-dependent lossy compression scheme.
- A truncated singular value decomposition (SVD) finds the best low-rank approximation to any matrix. PCA can be expressed in terms of the SVD.

Notation

Conventions:

- Small italic letters indicate either scalars or vectors. The ith element of vector a is given by $[a]_i$.
- Matrices are shown in capital letters, e.g. matrix A (with some exceptions, such as mean squared error MSE). The ith column of matrix A is often abbreviated as $a_i := [A]_{\cdot,i}$
- Random variables are shown with a boldface symbol, e.g. random variable $\mathbf{a}$ may have a realization a.
- An estimated quantity carries a hat, e.g. an estimate for quantity a is written $\hat{a}$.

List of symbols:

$\mathcal{I}$	indicator function, one or zero depending on whether its argument is true or false.
$\mathfrak{R}$	risk
$\mathfrak{L}$	loss function
$\mathbb{P}$	probability of an event
p	probability density
$\hat{p}$	estimate of probability density
cdf	cumulative distribution function
$\mathbb{E}$	expectation value
Var	variance
$\mathcal{X}$	input domain
$\mathcal{Y}$	output domain / set of all possible labels
$\mathfrak{n}$	number of observations
$\mathfrak{p}$	dimensionality of observations
$\mathfrak{c}$	number of classes
$\mathfrak{r}$	rank
$1^{a\times b}$	an $a \times b$ matrix filled with 1s
$I^{\mathfrak{p}}$	$\mathfrak{p} \times \mathfrak{p}$ identity matrix
$\mathcal{N}(\mu, \Sigma)$	normal distribution with mean μ and covariance matrix Σ
$k(\cdot)$	kernel function
k	a number, as in k-nearest neighbors or k-means clustering
$x_{1:\mathfrak{n}}$	set of samples $\{x_i \vert i = 1, \ldots, \mathfrak{n}\}$
MSE	mean squared error (in kernel density estimation)

$\mathfrak{E}$	error, residual
$\mathcal{P}_m, \mathcal{P}_d$	penalties for misclassification, doubt in a loss function
$\mathbb{L}_2$	l2-norm, Euclidean distance

Index

Bibliography

[1] Reference to be completed.

[2] M. Bach. http://michaelbach.de/ot/index.html.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? pages 217–235. Springer, 1999.

[4] L. Bottou and Y. Bengio. Convergence properties of the K-means algorithms. pages 585–592, 1995.

[5] Y. Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, 1995.

[6] J.H. Conway, N. J. A. Sloane, and E. Bannai. *Sphere packings, lattices, and groups*. Springer Verlag, 1999.

[7] J. H. Friedman. Regularized discriminant analysis. *Journal of the American Statistical Association*, 84(405):165–175, 1989.

[8] K. Fukunaga and L. D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Trans. Information Theory*, 21:32–40, 1975.

[9] G.H. Golub and C.F. Van Loan. *Matrix computations*. Johns Hopkins University Press, 1996.

[10] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *ICCV*, volume 2, pages 1458–1465, 2005.

[11] O.-J. Grüsser and M. Hagner. On the history of deformation phosphenes and the idea of internal light generated in the eye for the purpose of vision. *Documenta Ophthalmologica*, 74:57–85, 1990.

[12] E. H. P. A. Haeckel. *Generelle Morphologie der Organismen: allgemeine Grundzüge der organischen Formen-Wissenschaft, mechanisch begründet durch die von C. Darwin reformirte Decendenz-Theorie*. Berlin, 1866.

[13] F. A. Hamprecht and E. Agrell. Exploring a space of materials: spatial sampling design and subset selection. In J. N. Cawse, editor, *Experimental Design for Combinatorial and High Throughput Materials Development*, chapter 13. Wiley, New York, 2003.

[14] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, 2001.

[15] B. Jähne. *Digital image processing*. Springer, Heidelberg, 6th edition, 2007.

[16] H. R. Künsch, E. Agrell, and F. A. Hamprecht. Optimal lattices for sampling. *IEEE Transactions on Information Theory*, 51:634–647, 2005.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278–2324, 1998.

[18] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *ICCV*, 2005.

[19] T. Leung and J. Malik. Representing and recognizing the visual appearance of materials using three-dimensional textons. *International Journal of Computer Vision*, 43(1):29–44, 2001.

[20] S.E. Palmer. *Vision science: Photons to phenomenology*. MIT press, Cambridge, MA., 1999.

[21] D. W. Scott. *Multivariate Density Estimation*. Wiley, New York, 1992.

[22] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Monographs on Statistics and Applied Probability. Chapman & Hall, 1986.

[23] P. Simard, Y. LeCun, J. Denker, and B. Victorri. Transformation invariance in pattern recognition – tangent distance and tangent propagation. *Neural networks: tricks of the trade*, pages 549–550, 1998.

[24] M. Sugiyama, M. Krauledat, and K.R. Müller. Covariate shift adaptation by importance weighted cross validation. *The Journal of Machine Learning Research*, 8:985–1005, 2007.

[25] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, volume 591, pages 586–591, 1991.

[26] M. Varma and A. Zisserman. A statistical approach to material classification using image patch exemplars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2032–2047, 2008.