Mathematical Aspects in Machine Learning

FU Berlin, Summer 2017

Ralf Banisch

Contents

1 Introduction: What is Machine Learning?

Machine Learning emerged in the 1950ies and 1960ies as a subfield of computer science. According to a quote by the Machine Learning pioneer Arthur Samuel[1] in 1959 [Sam00], it involves

> *'The study of algorithms that give computers the ability to learn without being explicitly programmed.'*

This definition sounds appealing but a bit vague - what does 'having the ability to learn' mean precisely? We are used to the *cognitive* concept of learning - 'having the ability to learn' then means learning like a human. This puts machine learning firmly in the realm of artificial intelligence research, and indeed the two fields evolved closely together until the late 80ies. Things changed in the 90ies - Machine Learning evolved as a separate field and people settled for an *operational* rather then a *cognitive* definition[2]. One may quote Tom M. Mitchell [Mit97]:

> *'A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E.'*

There is a strong focus on automated methods, that learn without human intervention and purely from data. To illustrate how crucial this is, let's go back to a time in the late 90ies when there were *two* search engines for the internet (actually a few more): yahoo and google. Yahoo's approach was to create a phone book of the web: A curated data base of all the websites that could then be searched for queries. Google's approach

[1] Samuel actually wrote a program that could beat him at the game of checkers, after 8-10 hours of machine training time.

[2] To say it with Alan Turing's words: The question 'Can machines think?' should be replaced with the question 'Can machines do what we (as thinking entities) can do?' [Tur50]

was that of an automated algorithm that analyses the hyperlink structure between web pages. We all know what happened to yahoo, of course.

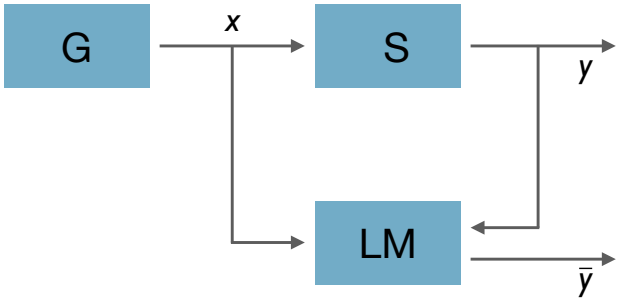

Figure 1.1 – The general scheme of supervised learning [VV98].

Supervised Learning. This is the 'classical' ML setup. It consists of a generator G who produces example data x from some set $\mathcal{X}$, a supervisor who applies an operator S to the data to produce the outcome $y = S(x) \in \mathcal{Y}$, and a learning machine LM that tries to imitate the supervisor's actions without knowledge of the operator S (see Figure 1.1). Typically, there are two phases:

1. Learning phase: LM learns a function $\overline{y} = f(x)$ from training data

   $$D = \{(x_1, y_1), \ldots, (x_l, y_l)\}.$$

2. Prediction phase: For unseen data points $x \in \mathcal{X}$, we now wish to predict the supervisor's response $S(x)$ with our learned function f. In other words, we require that $f(x)$ is a good approximation (in some sense) of $S(x)$.

In order to turn this setup into something we can do mathematics with, we would now have to specify what kind of operators the supervisor can apply, which classes of f's we consider and what it means that f is a good approximation for S (more on that later). There are many variants of this setup, in particular the learning could be done *offline* (no new training data once we are in the prediction phase), *online* (we get new training data while we are in the prediction phase), or even *active* (LM may choose $x \in \mathcal{X}$ and request the next training pair $(x, S(x))$).

Depending on the outcome space $\mathcal{Y}$, one typically distinguishes the following broad tasks:

1. **Classification.** The outcome space $\mathcal{Y}$ is discrete. In the simplest case, $y \in \{-1, 1\}$. That is, $\mathcal{X}$ consists of two classes, S returns the class label for each $x \in \mathcal{X}$, and the

task of LM is to learn a function which divides $\mathcal{X}$ correctly into its two classes (see Figure 1.2).

2. **Regression.** The outcome space $\mathcal{Y}$ continuous. In the simplest case, $y \in \mathbb{R}$. If the functions f that LM is allowed to consider are restricted to a certain class $\mathcal{C}$, then the task is to find the $f \in \mathcal{C}$ that fits the observed data D best (see Figure 1.2).

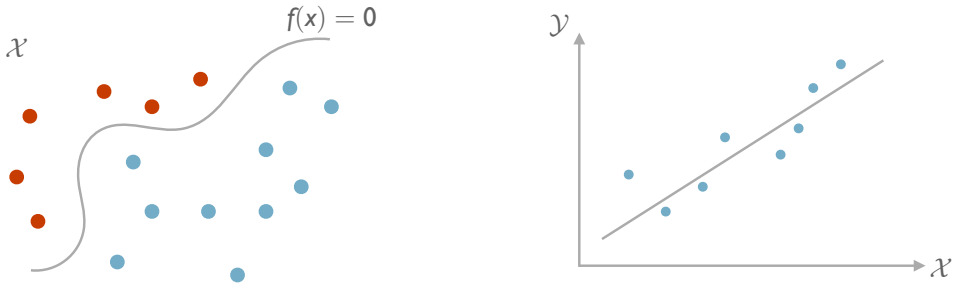

Figure 1.2 – Classification vs. regression.

Unsupervised learning. This is the more 'unorthodox' setup. We are *only* given input data $\{x_1, \ldots, x_l\} \in \mathcal{X}$. There is no supervisor (or at least we have no access to her answers). Typically, $\mathcal{X} \subset \mathbb{R}^n$ and n is large. The task is to 'learn something' about $\{x_1, \ldots, x_l\}$ - obviously this a much more fuzzy business, and different people will arrive at different answers depending on what they are looking for. Some examples of what to look for are

- What is the probability distribution function that generated $\{x_1, \ldots, x_l\}$?
- Is there a low-dimensional object $M \subset \mathcal{X}$ such that $\{x_1, \ldots, x_l\}$? What are the properties of M?

Unsupervised learning gained importance in the last 10-20 years because it is the situation that practitioners often face: There is a lot of data, nobody knows what it means, and one has to generate some knowledge about this data. This area of ML is very closely related to data mining. Manifold learning, dimension reduction, and clustering are buzzwords here.

Old field, but a lot of recent excitement. Machine Learning is an old field: Much of the theory dates back to 1950-1970. But there is no doubt that the interest in the field for much of the 20th century was minuscule compared to what it is in the year 2017, where machine learning is a subject which is 'in', perhaps even 'hyped'. In fact, according to the Gartner 2016 hype cycle[3], machine learning is at its peak of inflated expectations.

[3] http://www.gartner.com/newsroom/id/3412017

Given this enormous interest, two natural questions arise:

- Q1: Why is the interest so enormous now, and why did it take so long to get there?
- Q2: Should I really bother to get into the field now?

The first question is hard to answer, but some contributing factors are probably the following: First, until the late 80ies, ML was a subfield of Artificial Intelligence. And creating convincing AI's that can match the cognitive abilities of humans is extremely hard. Moreover, nobody is going to be impressed by your progress until you are basically done. In the 90ies, ML moved away from AI and towards making algorithms that get better and better at solving very specific problems. That eventually proved very successful. Second, there is of course much more computing power and much much more data available these days everywhere.

The answer to the second question depends a lot on one's own perspective. If you are thinking of becoming a practitioner in the industry, or you are thinking about going into research in some field other than ML that uses even moderate amounts of data, then you should *absolutely* bother about learning ML, because it will be part of your toolbox. If you are thinking about going into ML research, the answer is a lot more complex. My personal advice would be: You should bother because it's fun and rewarding, but you should also be smart about it. Just 'riding the wave of popularity', applying the latest algorithms to dish out nice applications, will probably not be enough - if that peak of inflated expectations passes, you'd likely be in trouble. Instead, focus on understanding the foundations and prepare for the peak to pass, because once things hit rock bottom, that's when the real understanding (and the real work) happens.

Understanding the mathematical foundations of ML is indeed the focus of this course. We will spend a lot of time with statistical learning theory, which dates back to 1970. We will look at algorithms and numerical examples, but they will be more often simple then sophisticated in order to keep the exposition clear. The second half of the course deals with kernel methods and unsupervised learning, and more recent work will feature there.

2 Statistical Learning Theory

Suggested Literature: [VV98], [DGL13], [BBL04].

In Statistical Learning Theory, one formalises the basic setup of supervised learning (see Figure 1.1) in probabilistic terms. We will see that the task of the learning machine - in the case of both classification and regression - is closely related to the general principle of *empirical risk minimization*. Much of this work is due to Vapnik [VV98]. Before diving into this theory though, let's look at one of the very first machine learning algorithms and understand its convergence.

2.1 Linear Classifiers Part I: The Perceptron

Suggested Literature: [Roj13] chapter 3 and 4.

We are concerned with a basic classification problem. Consider the following setup: We have training data

$$D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_l, y_l)\} \subset \mathbb{R}^n \times \{-1, 1\}.$$

Here and in the following, boldface symbols denote vectors $\boldsymbol{x} \in \mathbb{R}^n$, as opposed to scalars $y \in \mathbb{R}$. Our goal is to classify this data with a *linear classifier*. That is, we are looking for a hyperplane in $\mathbb{R}^n$ such that $y_i = +1$ if $\boldsymbol{x}_i$ is to the right and $y_i = -1$ if $\boldsymbol{x}_i$ is to the left of the hyperplane. In the ML context, the name *perceptron* was coined for such linear classifiers by Frank Rosenblatt in 1957:

Definition 2.1 (Perceptron, Frank Rosenblatt '57)**.** The perceptron with weight vector $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$ and threshold $\theta \in \mathbb{R}$ is the function

$$P_{\boldsymbol{w},\theta} : \mathbb{R}^n \to \{-1, +1\}, \quad P_{\boldsymbol{w},\theta}(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{n} w_i x_i - \theta\right)$$

where $\operatorname{sgn}(x) = +1$ for $x \geq 0$ and $\operatorname{sgn}(x) = -1$ for $x < 0$.

The perceptron is also the very simplest neural network one can imagine: It consists of just one neuron that takes $\boldsymbol{x} \in \mathbb{R}^n$ as input, applies the function $P_{\boldsymbol{w},\theta}$ with parameters $\boldsymbol{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, and returns ± 1 as output (see Figure 2.1). Despite all this fancy language, the perceptron all by itself is nothing impressive; it really is just a hyperplane that assigns the outcome $+1$ for points to the right and -1 for points to the left of the hyperplane. In order to solve the classification problem, we need an algorithm that can find the right parameters $\boldsymbol{w}$ and θ based on the training data D.

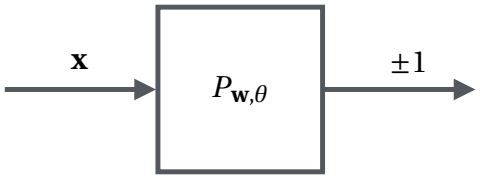

Figure 2.1 – The perceptron as a simple neural network.

A few remarks are in order:

- It suffices to consider $\theta = 0$, since otherwise we can extend the problem to $\mathbb{R}^{n+1}$ by introducing $w_{n+1} = -\theta$ and $x_{n+1} = 1$. Then

$$P_{\boldsymbol{w},\theta}(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{n+1} w_i x_i - \theta\right).$$

  We write $P_{\boldsymbol{w}} = P_{\boldsymbol{w},0}$.

- It doesn't matter if we take $\operatorname{sgn}(0) = +1$ or $\operatorname{sgn}(0) = -1$, as will become clear shortly.

Before moving on to the perceptron learning algorithm, it is worth checking if our goal of separating D by a hyperplane is actually achievable:

Definition 2.2. Two sets A and B of points in $\mathbb{R}^n$ are called *linearly separable* if $\boldsymbol{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ exist such that $\boldsymbol{w} \cdot \boldsymbol{x} \geq \theta$ for all $\boldsymbol{x} \in A$ and $\boldsymbol{w} \cdot \boldsymbol{x} < \theta$ for all $\boldsymbol{x} \in B$.

A and B are called *absolutely linearly separable* if $\boldsymbol{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ exist such that $\boldsymbol{w} \cdot \boldsymbol{x} > \theta$ for all $\boldsymbol{x} \in A$ and $\boldsymbol{w} \cdot \boldsymbol{x} < \theta$ for all $\boldsymbol{x} \in B$.

Perceptron learning requires absolute linear separability and not just linear separability. This is not a restriction when working with finite data, as the following proposition shows.

Proposition 2.3. If $A, B \subset \mathbb{R}^n$ are finite and linearly separable, then A and B are also absolutely linearly separable.

Proof. Since A and B are linearly separable, there are $\boldsymbol{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$ such that $\boldsymbol{w} \cdot \boldsymbol{x} \geq \theta$ for all $\boldsymbol{x} \in A$ and $\boldsymbol{w} \cdot \boldsymbol{x} < \theta$ for all $\boldsymbol{x} \in B$. Let us denote the corresponding hyperplane by $h_{\boldsymbol{w},\theta} = \{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{w} \cdot \boldsymbol{x} = \theta\}$. Since B is finite,

$$\operatorname{dist}(h_{\boldsymbol{w},\theta}, B) = \min_{\boldsymbol{x} \in B} \operatorname{dist}(h_{\boldsymbol{w},\theta}, \boldsymbol{x}) = \varepsilon > 0.$$

Now move $h_{\boldsymbol{w},\theta}$ by $\varepsilon/2$ in the direction of $-\boldsymbol{w}$. This absolutely separates A and B. □

2.1.1 Perceptron Learning

Suppose now that our classification goal is indeed achievable; that is, the sets $A = \{(\boldsymbol{x}_i, y_i) \in D | y_i = +1\}$ and $B = \{(\boldsymbol{x}_i, y_i) \in D | y_i = -1\}$ are (absolutely) linearly separable. We would like to learn the correct hyperplane, i.e. the weights $\boldsymbol{w} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, that separates A and B. This is achieved by the algorithm that follows, also proposed by Frank Rosenblatt in 1957. From now on, we set $\theta = 0$.

Algorithm 1 (Perceptron Learning)

start:	select $\boldsymbol{w}_0 \in \mathbb{R}^n$ at random;
	set $t := 0$;
test:	pick $(\boldsymbol{x}_i, y_i) \in D$ at random;
	if $y_i = +1$ and $P_{\boldsymbol{w}_t}(\boldsymbol{x}_i) = +1$, goto test;
	if $y_i = +1$ and $P_{\boldsymbol{w}_t}(\boldsymbol{x}_i) = -1$, goto add;
	if $y_i = -1$ and $P_{\boldsymbol{w}_t}(\boldsymbol{x}_i) = -1$, goto test;
	if $y_i = -1$ and $P_{\boldsymbol{w}_t}(\boldsymbol{x}_i) = +1$, goto subtract;
add:	set $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{x}_i$ and $t := t + 1$, goto test;
subtract:	set $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \boldsymbol{x}_i$ and $t := t + 1$, goto test.

In order to turn this into a fully-fledged algorithm, we need a termination condition, e.g. 'stop if you tested all points in D without updating $\boldsymbol{w}_t$'. We shall assume that such a termination condition has been added, the algorithm then outputs the last weight vector $\boldsymbol{w}_{t^*}$. Some further remarks are in order:

- It is not necessary to pick data points $(\boldsymbol{x}_i, y_i) \in D$ at random. One can also cycle through them deterministically. However, testing each data point once is not enough: When $\boldsymbol{w}_t$ is modified, then data points that were correctly classified by $\boldsymbol{w}_t$ might be misclassified by $\boldsymbol{w}_{t+1}$.
- There is a straightforward geometric interpretation: If $y_i = +1$ and the angle

between $\boldsymbol{w}_t$ and $\boldsymbol{x}_i$ is larger then 90°, $\boldsymbol{w}_t$ is rotated towards $\boldsymbol{x}_i$. If $y_i = -1$ and the angle between $\boldsymbol{w}_t$ and $\boldsymbol{x}_i$ is smaller then 90°, $\boldsymbol{w}_t$ is rotated away from $\boldsymbol{x}_i$. See also Figure 2.2.

- Convergence behaviour: Typically $\|\boldsymbol{w}_t\|$ grows with t. If $\|\boldsymbol{w}_t\| \gg \|\boldsymbol{x}_i\|$, then $\boldsymbol{w}_t \pm \boldsymbol{x}_i$ is almost equal to $\boldsymbol{x}_i$. That is, $\boldsymbol{w}_t$ changes a lot in the beginning, and as more and more data points are tested, the changes become smaller.

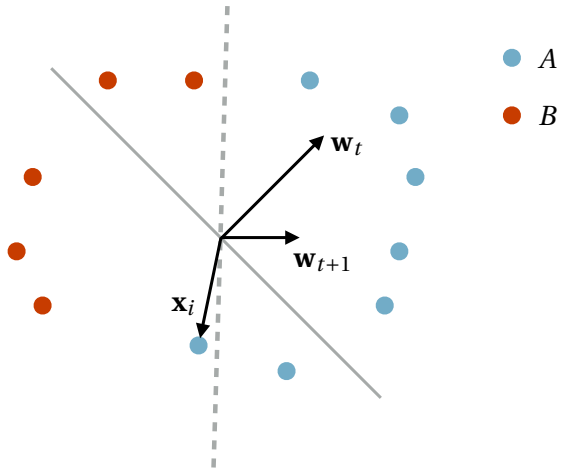

Figure 2.2 – The geometric interpretation of the perceptron learning algorithm. If the angle between $\boldsymbol{w}_t$ and $\boldsymbol{x}_i \in A$ is larger then 90°, then $\boldsymbol{w}_t$ is rotated towards $\boldsymbol{x}_i$.

We now state and prove a convergence theorem for algorithm 1.

Theorem 2.4 (Convergence of Perceptron Learning)**.** If the sets $A = \{(\boldsymbol{x}_i, y_i) \in D | y_i = +1\}$ and $B = \{(\boldsymbol{x}_i, y_i) \in D | y_i = -1\}$ are finite and linearly separable, then algorithm 1 updates $\boldsymbol{w}_t$ only a finite number of times. Moreover, the output $\boldsymbol{w}_{t^*}$ of algorithm 1 satisfies

$$P_{\boldsymbol{w}_{t^*}}(\boldsymbol{x}_i) = y_i \quad \forall (\boldsymbol{x}_i, y_i) \in D.$$

Proof. Since A and B are linearly separable and finite, there is $\boldsymbol{w}^* \in \mathbb{R}^n$ such that $\boldsymbol{w}^* \cdot \boldsymbol{x} > 0$ for all $\boldsymbol{x} \in A$ and $\boldsymbol{w}^* \cdot \boldsymbol{x} < 0$ for all $\boldsymbol{x} \in B$. To save some notation, we introduce $B^- = \{-\boldsymbol{x} | \boldsymbol{x} \in B\}$. Then $\boldsymbol{w}^*$ is such that $\boldsymbol{w}^* \cdot \boldsymbol{x} > 0$ for all $\boldsymbol{x} \in A \cup B^-$. In particular, perceptron learning with A and B is the same as perceptron learning with $A \cup B^-$ and all labels y_i equal to $+1$. Let

$$\delta = \min_{\boldsymbol{x} \in A \cup B^-} (\boldsymbol{w}^* \cdot \boldsymbol{x}) > 0.$$

We also normalize $\boldsymbol{w}^*$ so that $\|\boldsymbol{w}^*\| = 1$. Now suppose that at time t, the point $\boldsymbol{x}_i \in A \cup B^-$ was incorrectly classified (i.e. $\boldsymbol{w}_t \cdot \boldsymbol{x}_i \leq 0$), so that $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{x}_i$. We make two

observations. First,

$$\boldsymbol{w}^* \cdot \boldsymbol{w}_{t+1} = \boldsymbol{w}^* \cdot \boldsymbol{w}_t + \boldsymbol{w}^* \cdot \boldsymbol{x}_i \geq \boldsymbol{w}^* \cdot \boldsymbol{w}_t + \delta.$$

By induction, it follows that

$$\boldsymbol{w}^* \cdot \boldsymbol{w}_{t+1} \geq \boldsymbol{w}^* \cdot \boldsymbol{w}_0 + (t+1)\delta. \tag{2.1}$$

Second, with $M = \max_{\boldsymbol{x}_i \in A \cup B^-} \|\boldsymbol{x}_i\|^2$,

$$\|\boldsymbol{w}_{t+1}\|^2 = \|\boldsymbol{w}_t\|^2 + 2\boldsymbol{w}_t \cdot \boldsymbol{x}_i + \|\boldsymbol{x}_i\|^2 \leq \|\boldsymbol{w}_t\|^2 + M,$$

which follows from $\boldsymbol{w}^* \cdot \boldsymbol{x}_i \leq 0$. By induction, we get

$$\|\boldsymbol{w}_{t+1}\|^2 \leq \|\boldsymbol{w}_0\|^2 + (t+1)M. \tag{2.2}$$

Now let ρ_t be the angle between $\boldsymbol{w}_t$ and $\boldsymbol{w}^*$. By combining (2.1) and (2.2), we get

$$1 \geq \cos\rho_t = \frac{\boldsymbol{w}^* \cdot \boldsymbol{w}_t}{\|\boldsymbol{w}_t\|} \geq \frac{\boldsymbol{w}^* \cdot \boldsymbol{w}_0 + t\delta}{\sqrt{\|\boldsymbol{w}_0\|^2 + tM}}. \tag{2.3}$$

The RHS of (2.3) grows with $\sqrt{t}$, and is unbounded if t is unbounded. But since the RHS of (2.3) is also bounded from above by $1 \geq \cos\rho_t$, t must in fact by finite. Thus algorithm 1 terminates after a finite number t^* of iterations. By construction, $P_{\boldsymbol{w}_{t^*}}(\boldsymbol{x}_i) = y_i$ for all $(\boldsymbol{x}_i, y_i) \in D$, otherwise algorithm 1 would not have terminated. □

We end with some remarks on the convergence behaviour of perceptron learning.

- Under the conditions of theorem 2.4, t^* is always finite. However, it is easy to construct a dataset for which t^* can be very large in the worst case: Let $\boldsymbol{x}_i, \boldsymbol{x}_j \in A$ be such that the angle between with $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ is $180^\circ - \delta$. Then t^* is exponential in the margin δ in the worst case.
- If A and B are not linearly separable, then the perceptron learning algorithm does not terminate. We could still ask for a $\boldsymbol{w} \in \mathbb{R}^n$ which classifies the largest number of points in A and B correctly. This will be the goal in the next section.

2.2 Supervised Learning - Bayesian formulation

Suggested Literature: [VV98], [DGL13]

Recall the general supervised learning setup from figure 1.1. We will provide a mathematical formulation for this setup. In order to do that, we have to specify how the generator G produces data points $\boldsymbol{x} \in \mathcal{X}$, what types of operators $S : \mathcal{X} \to \mathcal{Y}$ the

supervisor is allowed to use, and what it means for the result of the learning machine $\overline{y}$ to be a good approximation of $y = S(x)$. We will work in the classification setting, that is, $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{-1, 1\}$. We will formulate the problem with the language of probability theory, see appendix A.

1. We model G by a random variable $X : \Omega \to \mathbb{R}^d$ with distribution μ:

$$\mu(A) = \mathbf{P}[X \in A], \quad A \in \mathcal{B}^d.$$

2. The outcome y will be modelled by another random variable $Y : \Omega \to -1, 1$. The supervisor S, upon seeing $X = x$, will draw the outcome $Y = 1$ according to the conditional probability distribution

$$\eta(x) = \mathbf{P}[Y = 1 | X = x], \quad x \in \mathbb{R}^d.$$

In the Baysian context, η is called the *posterior*, and μ is called the *prior* probability distribution. If μ has a density with respect to the Lebesgue measure on $\mathbb{R}^d$ (so that $\mu(dx) = \mu(x)dx$), then the pair (X, Y) has a joint probability density $p(x, y)$ on $\mathbb{R}^d \times \{-1, 1\}$ given by

$$\begin{aligned} p(x, 1) &= \eta(x)\mu(x), \\ p(x, -1) &= (1 - \eta(x))\mu(x). \end{aligned}$$

The goal of the learning machine LM is now to imitate the action of the supervisor, that is, the unknown conditional probability distribution function η, by a learned function $f : \mathbb{R}^d \to \{-1, 1\}$. Our probabilistic framework automatically gives us a measure for how 'good' this learned function is.

Definition 2.5. A function $f : \mathbb{R}^d \to \{-1, 1\}$ is called a classifier or decision function. The quantity $L(f) = \mathbf{P}[f(X) \neq Y]$ is called the *error probability* of f. Of particular interest is the *Bayes decision function*

$$f^*(x) = \begin{cases} 1 & \text{if } \eta(x) > 1/2, \\ -1 & \text{otherwise} \end{cases}$$

and the *Bayes probability of error* $L^* = L(f^*)$.

The Bayes decision function uses full knowledge about η, and is such it is typically not attainable by LM. We now show that the Bayes decision function is indeed optimal.

Theorem 2.6. For any classifier $f : \mathbb{R}^d \to \{-1, 1\}$, $L^* \leq L(f)$.

Proof. Given $X = x$, the conditional error probability of f may be expressed as

$$\begin{aligned}
&\mathbf{P}[f(X) \neq Y | X = x] \\
&= 1 - \mathbf{P}[f(X) = Y | X = x] \\
&= 1 - \{\mathbf{P}[Y = 1, f(X) = 1 | X = x] + \mathbf{P}[Y = -1, f(X) = -1 | X = x]\} \\
&= 1 - \left\{ I_{\{f(x)=1\}} \mathbf{P}[Y = 1 | X = x] + I_{\{f(x)=-1\}} \mathbf{P}[Y = -1 | X = x] \right\} \\
&= 1 - \left\{ I_{\{f(x)=1\}} \eta(x) + I_{\{f(x)=-1\}} (1 - \eta(x)) \right\}.
\end{aligned}$$

Thus for every $x \in \mathbb{R}^d$,

$$\begin{aligned}
&\mathbf{P}[f(X) \neq Y | X = x] - \mathbf{P}[f^*(X) \neq Y | X = x] \\
&= \eta(x) \left(I_{\{f^*(x)=1\}} - I_{\{f(x)=1\}} \right) + (1 - \eta(x)) \left(I_{\{f^*(x)=-1\}} - I_{\{f(x)=-1\}} \right) \\
&= (2\eta(x) - 1) \left(I_{\{f^*(x)=1\}} - I_{\{f(x)=1\}} \right) \\
&\geq 0.
\end{aligned}$$

The last line follows from the definition of f^*. Namely, $2\eta(x) - 1 > 0$ implies $f^*(x) = 1$, and $2\eta(x) - 1 \leq 0$ implies $f^*(x) = -1$. Now integrate both sides with respect to $\mu(dx)$ and the statement follows. □

A few remarks are in order:

- Theorem 2.6 shows that the Bayes rule f^* is the optimal classifier. Hence, if the posterior $\eta(x)$ is known for all $x \in \mathbb{R}^d$, then there is nothing to learn: The *LM* should use f^*. But usually η is not known so f^* is not accessible.

- Optimal decisions are not error free. The Bayes error L^* exactly quantifies the remaining uncertainty due to the fact that S chooses y non-deterministically.

- The posterior η actually minimizes the squared error when predicting Y with f:

  $$\mathbf{E}[(\eta(X) - Y)^2] \leq \mathbf{E}[(f(X) - Y)^2].$$

  We leave it as an exercise to show this.

A simple example. Suppose we wish to predict whether a student passes a course, based solely on the number of hours X studied per week. Let $Y = \pm 1$ mean the student passes or fails, respectively. Let us assume that $\eta(x) = x/(x + c)$ for some $c > 0$. This assures that η is monotonically increasing in x, which makes sense. The Bayes rule for

this problem becomes

$$f^*(x) = \begin{cases} +1 & \text{if } x > c, \\ -1 & \text{otherwise.} \end{cases}$$

In this problem, knowing the functional form of $\eta(x)$ and the value of $c > 0$ would therefore give us access to f^*. Computing the Bayes error L^* on the other hand requires additional knowledge of the distribution μ of X. Suppose first that $X = c$ with probability one - think of an army school where every student is forced to study c hours per week. Then

$$L^* = \mathbf{P}[f^*(X) \neq Y] = \mathbf{P}[f^*(X) \neq Y | X = c] = \mathbf{P}[Y \neq -1 | X = c] = \eta(c) = 1/2.$$

So in this case we actually have no way of making an informed decision - even the Bayes rule is not better then flipping a coin. This is because observing the outcome of X gives no additional information - $X = c$ with probability one anyway.

The situation improves if X is uniform on, say, $[0, 4c]$. Then

$$\begin{aligned} L^* &= \frac{1}{4c} \int_0^{4c} \mathbf{P}[f^*(X) \neq Y | X = x] dx \\ &= \frac{1}{4c} \int_0^{c} \mathbf{P}[Y \neq -1 | X = x] + \frac{1}{4c} \int_c^{4c} \mathbf{P}[Y \neq 1 | X = x] dx \\ &= \frac{1}{4c} \int_0^{c} \eta(x) dx + \frac{1}{4c} \int_c^{4c} (1 - \eta(x)) dx \\ &= \frac{1}{4c} \int_0^{c} \frac{x}{x+c} dx + \frac{1}{4c} \int_c^{4c} \frac{c}{x+c} dx \\ &= \frac{1}{4} \log \frac{5e}{4} \approx 0.306. \end{aligned}$$

2.2.1 Risk minimization

The distributions η and μ are usually not known, and so the Bayes classifier is intangible. What should the learning machine do instead? Suppose LM can choose classifiers ϕ from some function class $\mathcal{C}$. We could then look for a *class-optimal* function, that is, a classifier $\phi_{\mathcal{C}}^*$ such that

$$L(\phi_{\mathcal{C}}^*) = \inf_{\phi \in \mathcal{C}} L(\phi). \tag{2.4}$$

In theory, the LM could obtain $\phi_{\mathcal{C}}^*$ by minimizing $L(\phi)$ over the class $\mathcal{C}$ (assuming for the moment that such a minimum indeed exists within the class $\mathcal{C}$). While that sounds like a good idea, note that the error probability $L(\phi) = \mathbf{P}[\phi(X) \neq Y]$ is also an intangible quantity. We therefore cannot ever hope to solve the optimization problem (2.4).

Rather, LM needs some empirical rule for picking a function $\widehat{\phi}_l \in \mathcal{C}$ based on training data $D_l = \{(x_1, y_1), \ldots, (x_l, y_l)\}$, where the (x_i, y_i) are drawn independently from the joint distribution of (X, Y). Note the explicit index l on $\widehat{\phi}_l$ and D_l, indicating that the function picked will change if the amount of training data changes.

Suppose that we have such a rule. We can quantify how good $\widehat{\phi}_l$ is by using the empirical error and the Bayes rule:

$$0 \leq L(\widehat{\phi}_l) - L^* = \left(L(\widehat{\phi}_l) - \inf_{\phi \in \mathcal{C}} L(\phi) \right) + \left(\inf_{\phi \in \mathcal{C}} L(\phi) - L^* \right). \tag{2.5}$$

This splits up the error probability of $\widehat{\phi}_l$ into two parts: The first is the *estimation error*, which says how good our empirical classifier $\widehat{\phi}_l$ is compared to the best classifier in the class $\mathcal{C}$. The second is the *approximation error*, which says how good the class $\mathcal{C}$ is compared to the best possible classifier. See also Figure 2.3. The size of $\mathcal{C}$ is a compromise: If $\mathcal{C}$ is large then the approximation error is small, but the estimation error is probably large. If $\mathcal{C}$ is small, then the estimation error will be small but the approximation error will be large.

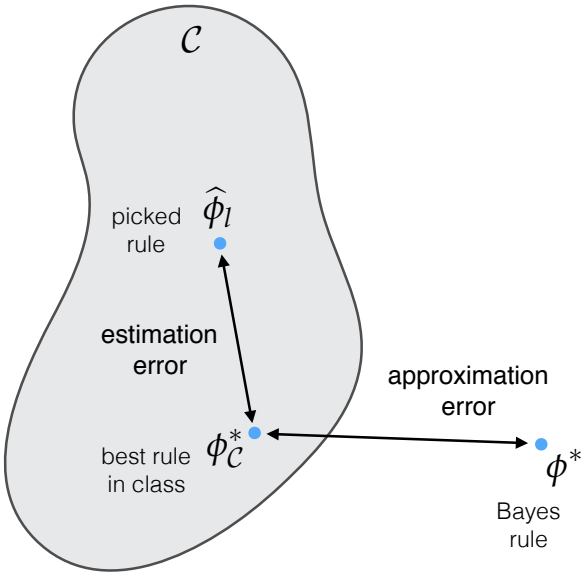

Figure 2.3 – Various errors in empirical classifier selection.

As an example, suppose that $\mathcal{C}$ is the class of all measurable decision functions. Then an empirically optimal classifier is

$$\widehat{\phi}_l(x) = \begin{cases} y_i & \text{if } x = x_i, \quad i = 1, \ldots, n \\ -1 & \text{otherwise.} \end{cases}$$

This classifier makes zero errors on D_l, but has no hope of performing well on unseen

data, so it is clearly not what we are looking for! This phenomenon is called *overfitting*. This raises two questions:

1. How should we choose $\mathcal{C}$? Should $\mathcal{C}$ grow with l?
2. Given $\mathcal{C}$ and $D_l = \{(x_1, y_1), \ldots, (x_l, y_l)\}$, what is a good rule to pick $\widehat{\phi}_l$?

We will leave the first question for later and now state an answer to the second question, which will keep us busy for quite a while:

Definition 2.7 (Empirical Risk)**.** For a classifier $\phi : \mathbb{R}^d \to \{-1, 1\}$ and training data $D_l = \{(x_1, y_1), \ldots, (x_l, y_l)\}$, the *empirical error probability* (or empirical risk) is defined as

$$\widehat{L}_l(\phi) = \frac{1}{l} \sum_{i=1}^{l} I_{\{\phi(x_i) \neq y_i\}}. \tag{2.6}$$

The empirically optimal rule is

$$\widehat{\phi}_l = \operatorname*{argmin}_{\phi \in \mathcal{C}} \widehat{L}_l(\phi). \tag{2.7}$$

A few remarks about this definition:

- The empirical risk counts the number of times the decision function ϕ fails on the training data.
- $\widehat{L}_l(\phi)$ is an estimator of $L(\phi)$. Indeed,

  $$\lim_{l \to \infty} \widehat{L}_l(\phi) = \lim_{l \to \infty} \frac{1}{l} \sum_{i=1}^{l} I_{\{\phi(x_i) \neq y_i\}} = \mathbf{E}\left[I_{\{\phi(X) \neq Y\}} \right] = \mathbf{P}[\phi(X) \neq Y] = L(\phi)$$

  with convergence in probability due to the weak law of large numbers.
- In contrast to (2.5), which is an *inaccessible* optimization problem, the optimization problem (2.7) is accessible by LM: It only involves the computation of quantities the learning machine knows explicitly. However, it might still be a difficult optimization problem computationally. If $|\mathcal{C}|$ is finite, then a brute force method can solve (2.7) in $\mathcal{O}(|\mathcal{C}|)$, which might be very large. In general, solving (2.7) is a problem in optimization theory that we leave for later.

The strategy of minimizing $\widehat{L}_l$ is called *empirical risk minimization*. It is quite intuitive: Just pick a function out of $\mathcal{C}$ that performs best on the training data. Our next goal will be to show that empirical risk minimization is indeed a very good idea, and that will be

quite some hard work. We leave the (potentially very serious!) computational problems that arise when one tries to actually perform the minimization (2.7) aside for now.

To demonstrate that empirical risk minimization is a good strategy, consider the case when $\mathcal{C}$ is finite and one of the classifiers in $\mathcal{C}$ has zero error probability. We start with a small but useful result.

Proposition 2.8. If $\min_{\phi\in\mathcal{C}} L(\phi) = 0$, then with probability one $\widehat{L}_l(\widehat{\phi}_l) = 0$, where $\widehat{\phi}_l$ is the empirically optimal rule as in (2.7).

Proof. Exercise. □

Proposition 2.8 says that if one of the classifiers in $\mathcal{C}$ has zero error probability, then with probability one we will be able to choose a classifier $\widehat{\phi}_l$ that has zero error on the training data. Of course that doesn't mean that $\widehat{\phi}_l$ makes zero errors on unseen samples. The error on unseen samples - or *generalization error* - is $L(\widehat{\phi}_l)$. Now we are ready for our first theorem from VC theory:

Theorem 2.9 (Vapnik and Chervonenkis 1974)**.** Assume $|\mathcal{C}| < \infty$ and $\min_{\phi\in\mathcal{C}} L(\phi) = 0$. Then for every l and $\varepsilon > 0$,

$$\mathbf{P}\left[L(\widehat{\phi}_l) > \varepsilon\right] \leq |\mathcal{C}|e^{-l\varepsilon}. \tag{2.8}$$

$$\mathbf{E}\left[L(\widehat{\phi}_l)\right] \leq \frac{1+\log|\mathcal{C}|}{l}. \tag{2.9}$$

where $\widehat{\phi}_l$ is the empirically optimal rule as in (2.7).

Proof. We start with the first inequality. Using proposition 2.8, we have

$$\mathbf{P}\left[L(\widehat{\phi}_l) > \varepsilon\right] \leq \mathbf{P}\left[\max_{\phi\in\mathcal{C}:\widehat{L}_l(\phi)=0} L(\phi) > \varepsilon\right] = \mathbf{E}\left[I_{\{\max_{\phi\in\mathcal{C}:\widehat{L}_l(\phi)=0} L(\phi)>\varepsilon\}}\right]. \tag{2.10}$$

The first inequality holds since $\widehat{L}_l(\widehat{\phi}_l) = 0$ with probability one, so maximizing with respect to all functions that have zero empirical error increases the probability. The second equality is due to $\mathbf{P}[A] = \mathbf{E}[I_A]$. Now

$$I_{\{\max_{\phi\in\mathcal{C}:\widehat{L}_l(\phi)=0} L(\phi)>\varepsilon\}} = \max_{\phi\in\mathcal{C}} I_{\{\widehat{L}_l(\phi)=0\}} I_{\{L(\phi)>\varepsilon\}} \tag{2.11}$$

since the LHS is the indicator function of the event "out of all the $\phi \in \mathcal{C}$ with zero empirical error, the largest $L(\phi)$ is larger than ε" whereas the RHS is the event "out of all $\phi \in \mathcal{C}$, there is at least one with both zero empirical error and $L(\phi) > \varepsilon$". Finally,

$$\mathbf{E}\left[\max_{\phi\in\mathcal{C}} I_{\{\widehat{L}_l(\phi)=0\}} I_{\{L(\phi)>\varepsilon\}}\right] \leq \sum_{\phi\in\mathcal{C}:L(\phi)>\varepsilon} \mathbf{P}\left[\widehat{L}_l(\phi) = 0\right] \leq |\mathcal{C}|(1-\varepsilon)^l \tag{2.12}$$

since the probability that no (x_i, y_i) pair falls in the set $\{(x, y) : \phi(x) \neq y\}$ is less then $(1-\varepsilon)^l$ if the probability of the set is larger then ε. The probability inequality of the theorem now follows by combining (2.10) - (2.12) and the simple inequality $1 - x \leq e^{-x}$.

To bound the expected error probability, note that for any $u > 0$,

$$\begin{aligned}
\mathbf{E}\left[L(\widehat{\phi}_l)\right] &= \int_0^\infty \mathbf{P}\left[L(\widehat{\phi}_l) > t\right] dt \\
&\leq u + \int_u^\infty \mathbf{P}\left[L(\widehat{\phi}_l) > t\right] dt \\
&\leq u + |\mathcal{C}| \int_u^\infty e^{-lt} dt \\
&= u + \frac{|\mathcal{C}|}{l} e^{-lu}.
\end{aligned}$$

Since u was arbitrary, we may choose it to minimize the obtained bound. The optimal choice is $u = \log |\mathcal{C}|/l$, which yields the desired result. □

2.2.2 Consistency

In the last section, we replaced a problem that we would like to but cannot solve - find the best rule ϕ in the given class $\mathcal{C}$ - with one that we can solve: Given a finite amount of data $D_l = \{(x_1, y_1), \ldots, (x_l, y_l)\}$ where (x_i, y_i) are i.i.d. realizations of the pair of random variables (X, Y), find a rule $\widehat{\phi}_l$ which is empirically optimal. Now a typical scenario is that D_l is not fixed - we might acquire more and more data. What can we expect about $\widehat{\phi}_l$ as $l \to \infty$?

Suppose we have a way of choosing a classification function $\widehat{\phi}_l(x, D_l)$ based on the observed data D_l, either by empirical risk minimization or some other rule that we agreed upon. The sequence $\left(\widehat{\phi}_l\right)_l$ is called a *classification rule*. Classification rules are rules for picking classifiers based on having seen an arbitrary number of training pairs. With a classification rule comes a sequence of error probabilities

$$L_l = \mathbf{P}[\widehat{\phi}_l(X, D_l) \neq Y | D_l].$$

Of course $L_l \geq L^*$ since the Bayes error L^* is optimal. It would be nice if $L_l \to L^*$ in some sense if $l \to \infty$. This is captured by the notion of consistency.

Definition 2.10. A classification rule $\left(\widehat{\phi}_l\right)_l$ is called *weakly consistent* with respect to the distribution of (X, Y) if

$$\lim_{l \to \infty} \mathbf{E}[L_l] = L^*.$$

The rule is called *strongly consistent* if $L_l \to L^*$ with probability one. It is called universally weakly (strongly) consistent if it is weakly (strongly) consistent with respect to any

distribution (X, Y).

Empirical risk minimization is one example of a classification rule, and theorem 2.9 shows that it is weakly consistent if $|\mathcal{C}| < \infty$ and $\min_{\phi \in \mathcal{C}} L(\phi) = 0$. To illustrate the difference between weak and strong consistency, consider the case $L^* = 0$. Then weak consistency means that the fraction of errors made by the learning machine goes to zero as more and more data is available. Strong consistency on the other hand means that the total number of errors is finite. Clearly, universal consistency is a very strong requirement since it makes absolutely no assumptions on the kinds of distributions (X, Y) one is facing. We will mostly be concerned with weak consistency.

2.3 Linear Classification Part II - Empirical Risk Minimization

We revisit the linear classification problem with the framework just developed. That is, $D_l = \{(x_1, y_1), \ldots, (x_l, y_l)\} \subset \mathbb{R}^d \times \{-1, 1\}$ and

$$\mathcal{C} = \{x \to w^T x + \theta : w \in \mathbb{R}^d, \theta \in \mathbb{R}\}$$

is the class of linear classifiers. In line of what has been said above, the LM should select an optimal classifier by minimizing the empirical error $\widehat{L}_l$ over this class. The class $\mathcal{C}$ has dimension $d + 1$ and obviously $|\mathcal{C}| = \infty$. So we can't apply theorem 2.9 in this situation. In any case, $\mathcal{C}$ is too large in order to be practical, since many member in $\mathcal{C}$ have the same performance in terms of $\widehat{L}_l$. We will now construct a smaller class $\mathcal{C}_l$ that is finite and such that $\min_{\phi \in \mathcal{C}_l} \widehat{L}_l(\phi)$ is very close to $\min_{\phi \in \mathcal{C}} \widehat{L}_l(\phi)$.

Suppose that X has a density w.r.t. the Lebesgue measure on $\mathbb{R}^d$. Now pick d arbitrary points $x_{i_1}, \ldots, x_{i_d}$ from the training set D_l. With probability one, these d points will be in general position and thus uniquely determine a hyperplane $w^T x + \theta = 0$ which goes through those points. To each such hyperplane, there are two classifiers

$$\phi_{\pm}(x) = \pm \operatorname{sgn}(w^T x + \theta).$$

Let $\mathcal{C}_l$ be the class of all classifiers thus constructed. Clearly, $|\mathcal{C}_l| = 2\binom{l}{d}$.

Proposition 2.11. With $\mathcal{C}_l$ as above, we have

$$\min_{\phi \in \mathcal{C}_l} \widehat{L}_l(\phi) \leq \min_{\phi \in \mathcal{C}} \widehat{L}_l(\phi) + \frac{d}{l}. \tag{2.13}$$

Proof. We leave the details as an exercise. The idea is that for every $\phi \in \mathcal{C}$, there is a $\widehat{\phi} \in \mathcal{C}_l$ such that the two decisions agree on all data points except possibly for the d data points used to define $\widehat{\phi}$. The (2.13) immediately follows. □

2.3.1 The linearly separable case revisited

Previously, linear separability was defined purely on the basis of the observed data. The issue with this definition is that we don't know if linear separability was really a property of the underlying problem or just an artefact of the data we happened to observe. With the Bayesian formulation of the problem in terms of the random variables $X, Y)$ at hand, it is now time to revisit this definition.

Definition 2.12. Let the random variable X take values in $\mathbb{R}^d$ and let $\mathcal{C} = \{x \to w^T x + \theta : w \in \mathbb{R}^d, \theta \in \mathbb{R}\}$ be the class of linear classifiers. We call the distribution (X, Y) linearly separable if $\min_{\phi \in \mathcal{C}} L(\phi) = 0$.

Definition 2.12 says that the classification problem modelled by the pair of random variables (X, Y) is linearly separable if there is a linear classifier with zero error probability. It follows by considering Proposition 2.8 that if (X, Y) is linearly separable, then the data is as well: $\min_{\phi \in \mathcal{C}} \widehat{L}_l(\phi) = 0$ with probability one. Equation (2.13) in turn implies

$$(X, Y) \text{ linearly separable } \Rightarrow \min_{\phi \in \mathcal{C}_l} \widehat{L}_l(\phi) \leq \frac{d}{l} \quad \text{w. prob. 1.}$$

Since the remaining error comes from the d points used to construct the minimizer $\widehat{\phi}_l = \operatorname{argmin}_{\phi \in \mathcal{C}_l} \widehat{L}_l(\phi)$, we can define an modified empirical risk $\widehat{L}_{l-d}$ as the empirical risk on the remaining $l - d$ points. For this modified empirical risk then, we have $\min_{\phi \in \mathcal{C}_l} \widehat{L}_{l-d}(\phi) = 0$ with probability one if (X, Y) is linearly separable. Now we are in a situation where Theorem 2.9 can be applied to give the following result.

Corollary 2.13. Assume that X has a density, and that the best linear classifier has zero probability of error. Let $\widehat{\phi}_l$ be found by minimizing the empirical risk $\widehat{L}_l$ over the class $\mathcal{C}_l$. Then for all $l > d$ and $\varepsilon \leq 1$,

$$\mathbf{P}\left[L(\widehat{\phi}_l) > \varepsilon\right] \leq 2\binom{l}{d} e^{-(l-d)\varepsilon}. \tag{2.14}$$

$$\mathbf{E}\left[L(\widehat{\phi}_l)\right] \leq \frac{d \log l + 2}{l - d}. \tag{2.15}$$

Proof. Noting that $|\mathcal{C}_l| = 2\binom{l}{d}$ and $\widehat{L}_{l-d}(\widehat{\phi}_l) = 0$ with probability one, this follows from Theorem 2.9 and $\binom{l}{d} \leq l^d$. □

The important thing to take home from this is that in the linearly separable case, the expected *generalization error* $L(\widehat{\phi}_l)$ (that is, the error on unseen samples) goes to zero with rate $\mathcal{O}(dl^{-1} \log l)$. Our classification rule is again weakly consistent. Note however that the work of actually computing the empirical minimizer $\widehat{\phi}_i$ by solving the optimization problem (2.7) grows with $\binom{l}{d}$, at least with a brute force method. Empirical risk minimization is expensive!

2.3.2 The general case

With $\mathcal{C}$ still the class of linear classifiers, let now

$$L_{\mathcal{C}} := \inf_{\phi \in \mathcal{C}} L(\phi) > 0,$$

in other words (X, Y) are not linearly separable. We are unfortunately not in a position to use Theorem 2.9 anymore, and indeed results are qualitatively different. We just state the result for this case.

Theorem 2.14. Assume that X has a density, and let $\widehat{\phi}_l$ be found by minimizing the empirical risk $\widehat{L}_l$ over the class $\mathcal{C}_l$. Then for all $l > d$ and $2d/l \leq \varepsilon \leq 1$,

$$\mathbf{P}\left[L(\widehat{\phi}_l) > L_{\mathcal{C}} + \varepsilon\right] \leq \left(2\binom{l}{d} + 1\right) e^{-l\frac{\varepsilon^2}{2} + 2d\varepsilon}. \tag{2.16}$$

$$\mathbf{E}\left[L(\widehat{\phi}_l) - L_{\mathcal{C}}\right] \leq \sqrt{\frac{2}{l}\left((d+1)\log l + 2(d+1)\right)}. \tag{2.17}$$

Proof. Can be found in [DGL13] chapter 4. □

When we compare the linearly separable ($L_{\mathcal{C}} = 0$) from the non linearly separable ($L_{\mathcal{C}} > 0$) case, we can see two differences:

1. The convergence rate is only $\mathcal{O}\left(\sqrt{dl^{-1}\log l}\right)$ when $L_{\mathcal{C}} > 0$ versus $\mathcal{O}(dl^{-1}\log l)$ when $L_{\mathcal{C}} = 0$. Classification is harder when error-free classification within the class is not possible.
2. In both cases, $L(\widehat{\phi}_l)$ converges in expectation value to $L_{\mathcal{C}}$, but in the linearly separable case $L_{\mathcal{C}} = L^* = 0$, whereas in the non linearly separable case it might very well be that $L_{\mathcal{C}} > L^*$. So empirical risk minimization is no longer (even weakly) consistent.

Note that both Corollary 2.13 and Theorem 2.14 make no assumption on (X, Y) other then that X should have a density. This assumption is necessary for the construction of the class $\mathcal{C}_l$, because we needed to assume that d randomly chosen points from D_l are in general position with probability one. With some care, this assumption too can be dropped. Thus the results here are *distribution free*, which is a typical feature of VC theory, which we will see in all its glory in the next section.

2.4 VC Theory

In this section, we leave linear classification behind and take a leap to the completely general case along the footsteps of the pioneering work of Vapnik and Chervonenkis

[VV98]. With this generality does come the need for further abstraction though, making this material a bit difficult to digest.

Let $\mathcal{C}$ now be a fixed class of decision functions which may or may not be finite. We will again pick a classifier by minimizing empirical risk

$$\widehat{\phi}_l = \operatorname*{argmin}_{\phi \in \mathcal{C}} \widehat{L}_l(\phi)$$

and we are again interested on obtaining bounds on the error probability $L(\widehat{\phi}_l)$. More specifically, we will ask the question (as before)

> "How large is the probability that our selected classifier performs significantly worse then the best classifier in the class?"

In other words, we want to obtain bounds on $\mathbf{P}[L(\widehat{\phi}_l) - L_{\mathcal{C}} > \varepsilon]$ where $L_{\mathcal{C}} = \inf_{\phi \in \mathcal{C}} L(\phi)$. In the previous section, bounds of this kind included a factor of $|\mathcal{C}|$. That clearly won't be helpful if $|\mathcal{C}| = \infty$. In that case, we need something else to measure the complexity of the decision problem. This leads to the concept of shatter coefficients, which we first introduce for collections of sets.

Definition 2.15 (shatter coefficient)**.** Let $\mathcal{A}$ be a collection of measurable sets. For $(z_1, \ldots, z_n) \subset \mathbb{R}^d$ let $N_{\mathcal{A}}(z_1, \ldots, z_n)$ be the number of different sets in

$$\{\{z_1, \ldots, z_n\} \cap A; A \in \mathcal{A}\}.$$

The nth *shatter coefficient* of $\mathcal{A}$ is

$$s(\mathcal{A}, n) = \max_{(z_1, \ldots, z_n) \subset \mathbb{R}^d} N_{\mathcal{A}}(z_1, \ldots, z_n).$$

That is, the shatter coefficient is the maximal number of different subsets of n points that can be picked out by the class of sets $\mathcal{A}$.

The shatter coefficients measure the richness of the class $\mathcal{A}$. Clearly, $s(\mathcal{A}, n) \leq 2^n$ since that is maximum number of different subsets of n points. If $N_{\mathcal{A}}(z_1, \ldots, z_n) = 2^n$ for some $(z_1, \ldots, z_n)$, then we say that $\mathcal{A}$ *shatters* $\{z_1, \ldots, z_n\}$. It turns out that the point where this starts to no longer hold is important, see below.

Example. Let $d = 2$ and let $\mathcal{A}$ be the class of rectangles in $\mathbb{R}^2$. The four points $z_1 = (0, 1)$, $z_2 = (1, 0)$, $z_3 = (0, -1)$ and $z_4 = (-1, 0)$ can be shattered by $\mathcal{A}$ since there are 16 rectangles that each pick one of the 16 different subsets of $\{z_1, z_2, z_3, z_4\}$ (see Figure 2.4). Five points however can never be shattered by $\mathcal{A}$: If the points are labelled as in the figure, then there is no rectangle $A \in \mathcal{A}$ such that $\{z_1, z_2, z_3, z_4\} \in A$ but $z_5 \notin A$.

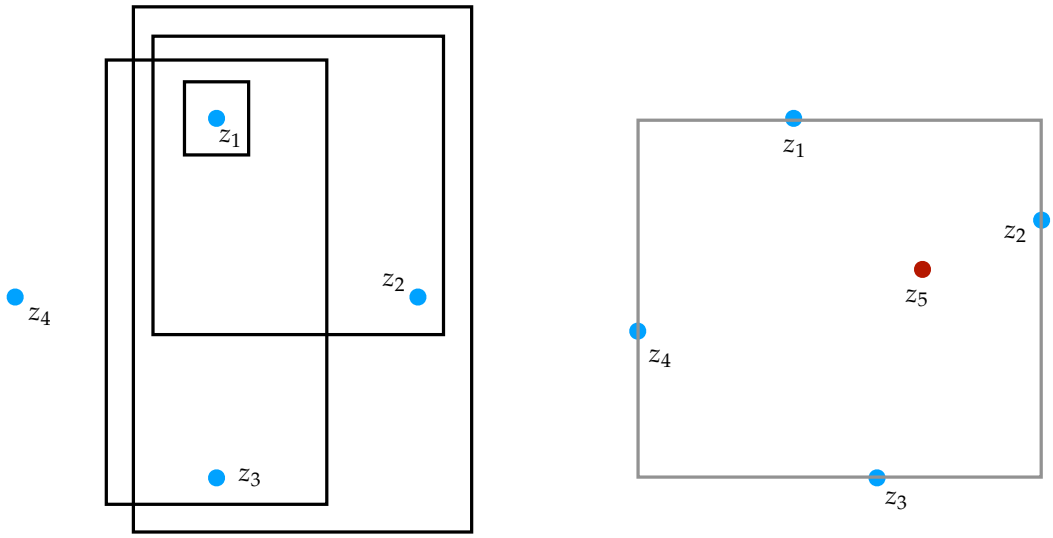

Figure 2.4 – Left: Four of the 16 rectangles used to shatter the four points $z_1 = (0,1)$, $z_2 = (1,0)$, $z_3 = (0,-1)$ and $z_4 = (-1,0)$ in the plane. Right: Five points in general position can not be shattered by rectangles since there is no rectangle $A \in \mathcal{A}$ such that $\{z_1, z_2, z_3, z_4\} \in A$ but $z_5 \notin A$.

Definition 2.16 (VC dimension)**.** Let $\mathcal{A}$ be a collection of sets, $|\mathcal{A}| \geq 2$. The VC dimension $V_{\mathcal{A}}$ of $\mathcal{A}$ is the largest integer $k \geq 1$ for which $s(\mathcal{A}, k) = 2^k$. If $s(\mathcal{A}, k) = 2^k$ for all $k \geq 1$, then $V_{\mathcal{A}} = \infty$.

Examples.

- Let $\mathcal{A}$ be the collection of all halflines $(-\infty, x]$, $x \in \mathbb{R}$. Then $s(\mathcal{A}, 2) = 3 < 2^2$ since for any two points $z_1 < z_2 \in \mathbb{R}$, there is no halfline $A \in \mathcal{A}$ such that $z_2 \in A$ in $z_1 \notin A$. Thus the VC dimension of the collection of all halflines is $V_{\mathcal{A}} = 1$.
- Let $\mathcal{A}$ be the collection of all rectangles in $\mathbb{R}^d$. Then $V_{\mathcal{A}} = 2d$. We showed this for $d = 2$ above, and the argument easily generalizes.

Suppose now that $z_1, \ldots, z_n$ are i.i.d. realizations of a random variable $Z : \Omega \to \mathbb{R}^d$ with distribution $\nu(A) = \mathbf{P}[Z \in A]$. How close is the *empirical distribution*

$$\nu_n(A) = \frac{1}{n} \sum_{i=1}^{n} I_{\{z_i \in A\}},$$

which is based on the samples $\{z_1, \ldots, z_n\}$, to the actual distribution ν of the random variable Z? The following theorem is central to VC theory.

Theorem 2.17 (Vapnik & Chervonenkis 1971)**.** For any probability measure ν and any class $\mathcal{A}$, and for any $n > 0$ and $\varepsilon > 0$,

$$\mathbf{P}\left[\sup_{A\in\mathcal{A}} |\nu_n(A) - \nu(A)| > \varepsilon\right] \leq 8s(\mathcal{A}, n)e^{-n\varepsilon^2/32}$$

Proof. In [DGL13], chapter 12. □

To appreciate this result, consider first what the central limit theorem has to tell us about the convergence of ν_n. For a fixed $A \in \mathcal{A}$, the central limit theorem gives convergence in distribution as $n \to \infty$:

$$\sqrt{n}(\nu_n(A) - \nu(A)) \to N(0, \sigma_A^2), \qquad \sigma_A^2 = \mathrm{Var}[I_{\{Z\in A\}}] = \nu(A)(1 - \nu(A)).$$

In other words the random variable $\nu_n(A)$ has mean $\nu(A)$ and variance $\mathcal{O}(\sqrt{n})$ for large n. Moreover, by using Hoeffding's inequality (Hoeffding 1963), we can characterize the probability of a large deviation of $\nu_n(A)$ from its mean $\nu(A)$:

$$\mathbf{P}[|\nu_n(A) - \nu(A)| > \varepsilon] \leq 2e^{-2n\varepsilon^2}.$$

The inequality in Theorem 2.17 is now the *uniform* version of this large deviation-type inequality. This uniform version is much harder to prove! If $|\mathcal{A}| < \infty$ then we get the following "easy" version of Theorem 2.17 directly from Hoeffding's inequality:

$$\mathbf{P}\left[\sup_{A\in\mathcal{A}} |\nu_n(A) - \nu(A)| > \varepsilon\right] \leq 2|\mathcal{A}|e^{-n\varepsilon^2/32}.$$

Again, this is not helpful if $|\mathcal{A}| = \infty$: The shatter coefficient $s(\mathcal{A}, n)$ plays the role of $|\mathcal{A}|$ if $|\mathcal{A}| = \infty$.

2.4.1 Classifier Selection

Theorem 2.17 talks about uniform deviations of relative frequencies from probabilities. What does this have to do with classifier selection? Let us now make this connection. So let $\mathcal{C}$ be a class of decision functions $\phi : \mathbb{R}^d \to \{-1, 1\}$. For every $\phi \in \mathcal{C}$, we define the "failure set"

$$A_\phi := \{\{x : \phi(x) = 1\} \times \{-1\}\} \bigcup \{\{x : \phi(x) = -1\} \times \{+1\}\} \subset \mathbb{R}^d \times \{-1, 1\}.$$

We also define the collection of failure sets $\mathcal{A} = \{A_\phi : \phi \in \mathcal{C}\}$. Then the nth shatter coefficient of the class $\mathcal{C}$ is

$$S(\mathcal{C}, n) = s(\mathcal{A}, n)$$

and the VC dimension of $\mathcal{C}$ is

$$V_{\mathcal{C}} = V_{\mathcal{A}}.$$

This prepares us for the following theorem, which treats the classification problem in full generality.

Theorem 2.18. For any class $\mathcal{C}$ of decision functions and any distribution (X, Y), we have

$$\mathbf{P}\left[\sup_{\phi\in\mathcal{C}} |\widehat{L}_l(\phi) - L(\phi)| > \varepsilon\right] \leq 8S(\mathcal{C}, l)e^{-l\varepsilon^2/32} \tag{2.18}$$

and for the minimizer $\widehat{\phi}_l$ of the empirical risk $\widehat{L}_l$,

$$\mathbf{P}\left[L(\widehat{\phi}_l) - \inf_{\phi\in\mathcal{C}} L(\phi) > \varepsilon\right] \leq 8S(\mathcal{C}, l)e^{-l\varepsilon^2/128}. \tag{2.19}$$

Proof. We note that $A_\phi \subset \mathbb{R}^d \times \{-1, 1\}$ is the set of combinations (x, y) where ϕ misclassifies x. Thus the probability that $\phi(X) \neq Y$ is the probability that $(X, Y) \in A_\phi$:

$$L(\phi) = \mathbf{P}[\phi(X) \neq Y] = \mathbf{P}\left[\{\phi(X) = 1 \cap Y = -1\} \bigcup \{\phi(X) = -1 \cap Y = 1\}\right] = \mathbf{P}[A_\phi].$$

If we denote by ν the probability measure induced by the random variable $Z = (X, Y)$, then $L(\phi) = \nu(A_\phi)$ follows. Further,

$$\widehat{L}_l(\phi) = \frac{1}{l}\sum_{i=1}^{l} I_{\{\phi(x_i)\neq y_i\}} = \frac{1}{l}\sum_{i=1}^{l} I_{\{(x_i,y_i)\in A_\phi\}} = \nu_l(A_\phi)$$

is the corresponding empirical measure. Equation (2.18) now follows from Theorem 2.17. For (2.19), note that

$$\begin{aligned} L(\widehat{\phi}_l) - \inf_{\phi\in\mathcal{C}} &= L(\widehat{\phi}_l) - \widehat{L}_l(\widehat{\phi}_l) + \widehat{L}_l(\widehat{\phi}_l) - \inf_{\phi\in\mathcal{C}} \\ &\leq L(\widehat{\phi}_l) - \widehat{L}_l(\widehat{\phi}_l) + \sup_{\phi\in\mathcal{C}} \left|\widehat{L}_l(\phi) - L(\phi)\right| \\ &\leq 2\sup_{\phi\in\mathcal{C}} \left|\widehat{L}_l(\phi) - L(\phi)\right|. \end{aligned}$$

□

Theorem 2.18 is a completely distribution free performance result of empirical risk minimization! This is quite astonishing, since distributions can be very strange. As in

Theorem 2.9, we can also easily obtain a bound for the expected value of $L(\widehat{\phi}_l)$:

$$\mathbf{E}\left[L(\widehat{\phi}_l)\right] - \inf_{\phi\in\mathcal{C}} L(\phi) \leq 16\sqrt{\frac{\log S(\mathcal{C},l)+4}{2l}}. \tag{2.20}$$

From here the problem is purely combinatorial: One has to estimate the shatter coefficients $S(\mathcal{C},l)$ for the classes $\mathcal{C}$ one is interested in. Some examples:

- **Finite VC dimension**. If $2 < V_{\mathcal{C}} < \infty$, then one can show that $S(\mathcal{C},l) \leq l^{V_{\mathcal{C}}}$. Equation (2.20) then becomes

  $$\mathbf{E}\left[L(\widehat{\phi}_l)\right] - \inf_{\phi\in\mathcal{C}} L(\phi) \leq 16\sqrt{\frac{V_{\mathcal{C}}\log l+4}{2l}}.$$

- **Infinite VC dimension.** If $V_{\mathcal{C}} = \infty$, then $S(\mathcal{C},l) = 2^l$ for all l, and consequently

  $$\mathbf{E}\left[L(\widehat{\phi}_l)\right] - \inf_{\phi\in\mathcal{C}} L(\phi) \leq 8\sqrt{2\log 2+\frac{8}{l}}.$$

  This bound is, of course, completely useless since an expected error of $1/2$ is already as bad as random guessing. $V_{\mathcal{C}} = \infty$ is not a good idea for classification.

- **Linear Classification**. Suppose $\mathcal{C}$ is the class of linear classifiers in $\mathbb{R}^d$. Then one can show that $V_{\mathcal{C}} = d+1$ (exercise[1]). Equation (2.20) becomes

  $$\mathbf{E}\left[L(\widehat{\phi}_l)\right] - \inf_{\phi\in\mathcal{C}} L(\phi) \leq 16\sqrt{\frac{(d+1)\log l+4}{2l}}.$$

  One should now compare this with Theorem 2.14! The two results essentially agree up to a factor of 8. The bound produced by Theorem 2.14 is tighter then the bound produced by the general theorem 2.18. That shouldn't be surprising: Theorem 2.14 only dealt with the linear classification problem and a very specific class of classifiers $\mathcal{C}_l$.

[1] In general, it is true that if $\mathcal{C}$ is a vector space of dimension k then $V_{\mathcal{C}} \leq k+1$.

3 Kernel Methods

Suggested Literature: [SS98]

Recall our basic setup: We have training data $x_1, \ldots, x_l \in \mathcal{X} \subset \mathbb{R}^d$ and labels $y_1, \ldots, y_l$. So far, we have used the data as it is, but this is not necessary (and in many cases not a good idea). Instead, one could think of transforming the data first using a *feature map*

$$\Phi : \mathcal{X} \to F.$$

Here, the *feature space* F is an arbitrary vector space that we will equip with a scalar product in a moment. There is a great deal of flexibility in choosing Φ. Machine Learning has its own subfield devoted to this called *feature engineering*. Usually one tries to engineer Φ such that the problem is simpler in Feature space, figure 3.1 shows an example. Here, it was possible to treat a non-linearly separable classification problem with a linear method that works in a feature space F where the problem is linearly separable. This will be indeed our strategy in this chapter - solve non-linear problems in $\mathcal{X}$ by mapping them to a feature space F where the problem becomes linear. In very high dimensional spaces almost everything looks linear, so for "rich enough" feature maps we can expect this strategy to work generically.

3.1 Reproducing kernel Hilbert spaces

There is a problem with our strategy: It is difficult to deal with high dimensional spaces explicitly. But there is a trick that comes to our rescue. Let $\langle \cdot, \cdot \rangle$ be the standard Euclidean inner product on F. For the example in Figure 3.1, we may compute it explicitly:

$$\langle \Phi(x), \Phi(x') \rangle = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2 \right)^T \left(x_1'^2, \sqrt{2} x_1' x_2', x_2'^2 \right) = (x^T x')^2 =: k(x, x').$$

What happens here is that there is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, defined as $k(x, x') = (x^T x')^2$, that allows us to compute the inner product $\langle \cdot, \cdot \rangle$ directly in $\mathcal{X}$ (and not in the

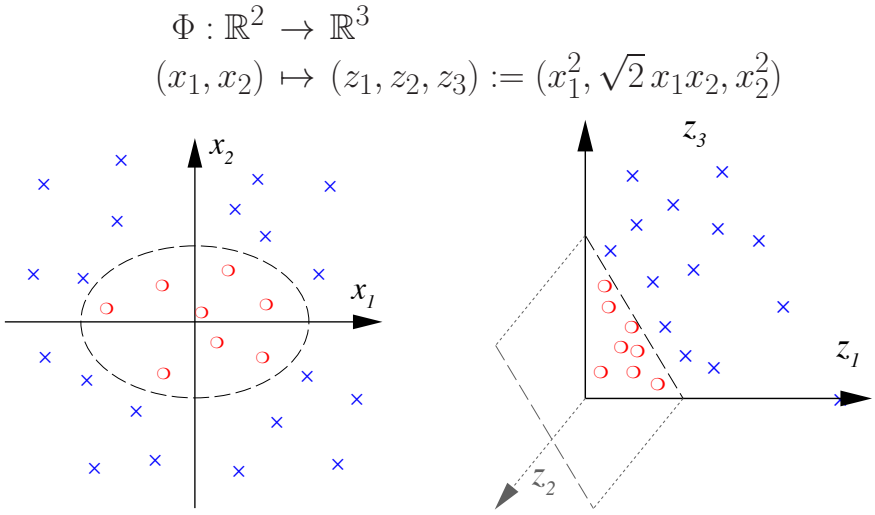

Figure 3.1 – A feature map can transform a non-linearly separable classification problem (left) into a linearly separable one in a larger feature space (right).

high dimensional space F). This is the so-called 'kernel trick', and the function k is called a *kernel*. Since k 'reproduces' a scalar product, it has some special properties:

Definition 3.1 (psd kernel)**.** A function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a *symmetric positive semi-definite (psd) kernel* if

- k is symmetric: $k(x, x') = k(x', x)$,
- k is positive semi-definite: $\sum_{i,j=1}^{m} a_i a_j k(x_i, x_j) \geq 0$ for any $x_1, \ldots, x_m \in \mathcal{X}$, $a_1, \ldots, a_m \in \mathbb{R}$ and $m \in \mathbb{N}$.

It turns out that the kernel trick works in general:

Lemma 3.2. Let F be a Hilbert space with inner product $\langle \cdot, \cdot \rangle$. For any map $\Phi : \mathcal{X} \to F$ there is a psd kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that k is reproducing:

$$\langle \Phi(x), \Phi(x') \rangle = k(x, x') \quad \forall x, x' \in \mathcal{X}. \tag{3.1}$$

Proof. Define k by (3.1). Then k is symmetric since $\langle \cdot, \cdot \rangle$ is symmetric, and

$$\sum_{i,j=1}^{m} a_i a_j k(x_i, x_j) = \sum_{i,j=1}^{m} a_i a_j \langle \Phi(x_i), \Phi(x_j) \rangle = \left\langle \sum_i a_i \Phi(x_i), \sum_j a_j \Phi(x_j) \right\rangle \geq 0$$

since $\langle \cdot, \cdot \rangle$ is bilinear and positive definite. □

Lemma 3.2 says that we can start with any feature map $\Phi : \mathcal{X} \to F$, endow F with an inner product and then we can find a psd kernel that reproduces this inner product. More interestingly, the converse is also true: We can *start* with a psd kernel k and then construct a Hilbert space for which this kernel is reproducing. This is the famous reproducing kernel Hilbert space (RKHS).

Theorem 3.3 (Moore, Aronszajn)**.** Let $\mathcal{X}$ be a nonempty set and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ a psd kernel. Then there exists a unique Hilbert space $(\mathcal{H}, \langle \cdot, \cdot \rangle)$ of functions on $\mathcal{X}$ and a map $\Phi : \mathcal{X} \to \mathcal{H}$ such that

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \quad \forall x, x' \in \mathcal{X}. \tag{3.2}$$

The space $\mathcal{H}$ is called the *reproducing kernel Hilbert space* (RKHS) and (3.2) the *reproducing property*.

Proof. The proof is constructive, we outline the main steps:

1. For any $x \in \mathcal{X}$, define the linear functional $\Phi(x) : \mathcal{X} \to \mathbb{R}$ by $\Phi(x) = k(x, \cdot)$.
2. Let $\mathcal{H}_0$ be the linear span of $\{\Phi(x) : x \in \mathcal{X}\}$. That is,

   $$f(\cdot) = \sum_{i=1}^{m} a_i k(x_i, \cdot) \in \mathcal{H}_0 \quad \forall\, x_i \in \mathcal{X}, a_i \in \mathbb{R}, m \in \mathbb{N}.$$

3. Define an inner product on $\mathcal{H}_0$ by

   $$\left\langle \sum_{j=1}^{n} b_j \Phi(y_j), \sum_{i=1}^{m} a_i \Phi(x_i) \right\rangle = \sum_{i=1}^{m} \sum_{j=1}^{n} a_i b_j k(y_j, x_i).$$

   This is a well-defined inner product due to the psd properties of k (we leave this for the reader to check).

4. Let $\mathcal{H}$ be the completion of $\mathcal{H}_0$ with respect to the inner product $\langle \cdot, \cdot \rangle$. $\mathcal{H}$ contains all Cauchy sequences

   $$f(x) = \sum_{i=1}^{\infty} a_i k(x_i, x)$$

   such that $\langle f, f \rangle < \infty$.

5. Check the reproducing property:

   $$\langle f, \Phi(x) \rangle = \left\langle \sum_{i=1}^{\infty} a_i \Phi(x_i), \Phi(x) \right\rangle = \sum_{i=1}^{\infty} a_i k(x_i, x) = f(x).$$

   In particular, for $f = \Phi(x')$ we have $\langle \Phi(x'), \Phi(x) \rangle = k(x, x')$.

6. To show uniqueness, let $\mathcal{G}$ be another Hilbert space for which k is reproducing, that is:

$$\langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}} = k(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{G}}.$$

By linearity, $\langle \cdot, \cdot \rangle_{\mathcal{H}} = \langle \cdot, \cdot \rangle_{\mathcal{G}}$ on $\mathcal{H}_0$. In other words, $\mathcal{H}_0 \subset \mathcal{G}$, and since $\mathcal{G}$ is complete, $\mathcal{H} \subset \mathcal{G}$. Now for an arbitrary $f \in \mathcal{G}$ let $f = f_{\mathcal{H}} + f_{\mathcal{H}^\perp}$ where $f_{\mathcal{H}} \in \mathcal{H}$ and $f_{\mathcal{H}^\perp} \in \mathcal{H}^\perp$. Due to the reproducing property

$$f(x) = \langle f, \Phi(x) \rangle = \langle f_{\mathcal{H}}, \Phi(x) \rangle = f_{\mathcal{H}}(x)$$

thus $f_{\mathcal{H}^\perp} = 0$ and $\mathcal{H} = \mathcal{G}$.

□

We name a few common kernels:

- linear kernel: $k(x, x') = x^T x'$. The feature map Φ is the identity.
- polynomial kernel: $k(x, x') = (x^T x + c)^n$. The feature map consists of all monomials of degree up to n. $\dim \Phi(X) < \infty$.
- gaussian kernel: $k(x, x') = \exp(-\sigma^{-1} \|x - x'\|^2)$. The feature map is very hard to construct explicitly. $\dim \Phi(X) = \infty$.

A few remarks are in order.

- any algorithm that only requires inner products of vectors $x_1, \ldots, x_l \in \mathbb{R}^d$ can be "kernelized" by replacing $x_i^T x_j$ with $k(x_i, x_j)$, which implicitly reproduces the algorithm in feature space. We do not need to know the feature map Φ explicitly for this. This procedure leads to kernel-PCA, kernel-kmeans, the kernel-Perceptron, and many other algorithms (two more examples of kernelization are in what follows).
- the feature space $\Phi(\mathcal{X})$ is usually high dimensional and not explicitly known.

3.2 Kernelization

We discuss two examples of the "kernelization" procedure outlined above. The idea in both examples is to solve a linear problem in $\mathcal{X}$ first and then promote the resulting algorithm to feature space by kernelization. The result is an algorithm that is linear in feature space but non linear in $\mathcal{X}$.

Before we begin, let us briefly pick things up where we left them at the end of chapter 2. Recall that towards the end of it we discussed bounds of the form[1]

$$\text{test error} \leq \text{training error} + \sqrt{\frac{V_{\mathcal{C}} \log l}{l}}.$$

Here, the 'test error' is the expected error on future samples $L(\phi) = \mathbf{E}[\phi(X) \neq Y]$, the 'training error' is the empirical risk $\widehat{L}_l$ on the training data and the VC dimension $V_{\mathcal{C}}$ was a measure for the complexity of the function class $\mathcal{C}$. Our strategy has been to minimize the empirical risk

$$\min_{\phi \in \mathcal{C}} \widehat{L}_l(\phi) = \min_{\phi \in \mathcal{C}} \frac{1}{l} \sum_{i=1}^{l} r(y_i, \phi(x_i)), \qquad r(y_i, \phi(x_i) = I_{\{y_i \neq \phi(x_i)\}}$$

which is an average of a loss function $r(y_i, \phi(x_i))$ over the training data. In our case, $r(y_i, \phi(x_i))$ was simply the 'zero-one loss function' which is zero if prediction was correct and one otherwise.

The presence of the VC dimension in the error bound above suggests that simple functions ϕ should be favoured over more complex ones. To achieve this, one could ask to minimize

$$\frac{1}{l} \sum_{i=1}^{l} r(y_i, \phi(x_i)) + \text{penalty}(\phi) \tag{3.3}$$

instead, with a penalty term that encourages simple functions. This strategy is called *structural risk minimization*.

3.2.1 Kernel ridge regression

We specialize (3.3) to the situation of a regression problem: $y_i \in \mathbb{R}$, $\phi \in \mathcal{C} = \mathcal{H}$ is a Hilbert space of functions, and $r(y_i, \phi(x_i)) = (y_i - \phi(x_i))^2$ is the so-called squared loss. The task is to minimize

$$\frac{1}{l} \sum_{i=1}^{l} (y_i - \phi(x_i))^2 + \tilde{\lambda} \|\phi\|_{\mathcal{H}}^2$$

over all $\phi \in \mathcal{H}$. This is called *ridge regression*, the penalty term $\|\phi\|_{\mathcal{H}}^2$ is the *Tikhonov regularization* and penalizes overfitting. In line with our strategy, we first consider the special case of linear ridge regression, that is, $\phi(x) = w^T x$ with $w \in \mathbb{R}^d$. The

[1] This is essentially the first part of Theorem 2.18 with an argument similar to that leading to (2.20), ommiting all constants.

minimization problem is now

$$\min_{w\in\mathbb{R}^d}\left\{\sum_{i=1}^{l}(y_i - w^T x_i)^2 + \lambda\|w\|^2\right\} \tag{3.4}$$

with $\lambda = \tilde{\lambda} l$. Although it is not necessary here, we will solve this problem with a detour in constrained optimization. First, (3.4) is reformulated as

$$\min_{w\in\mathbb{R}^d, z\in\mathbb{R}^l}\left\{\sum_{i=1}^{l} z_i^2 + \lambda\|w\|^2\right\}, \qquad z_i = y_i - w^T x_i. \tag{3.5}$$

This constrained optimization problem is solved with the method of Lagrange multipliers. The method consists of noticing that (3.5) is equivalent to the saddle point problem

$$\max_{\alpha}\min_{w,z}\mathcal{L}(w,z,\alpha)$$

with the Lagrangian

$$\mathcal{L}(w,z,\alpha) = \frac{1}{2}\sum_{i=1}^{l} z_i^2 + \frac{1}{2}\lambda\|w\|^2 + \sum_{i=1}^{l}\alpha_i(x_i^T w - y_i - z_i).$$

What we have done is to introduce a Lagrange multiplier α_i for each of the constraints in (3.5). Maximization of the Lagrangian over α enforces the constraints (just take derivatives to check this). Now lets solve the inner problem:

$$\begin{aligned} 0 = \frac{\partial\mathcal{L}}{\partial z_i} &= z_i - \alpha_i && \Rightarrow z^* = \alpha \\ 0 = \frac{\partial\mathcal{L}}{\partial w} &= \lambda w + \sum_i \alpha_i x_i && \Rightarrow w^* = -\frac{1}{\lambda}\sum_{i=1}^{l}\alpha_i x_i \end{aligned}$$

Hence, with the $l \times l$ matrix $K_{ij} = x_i^T x_j$, we have

$$\begin{aligned} \mathcal{L}(w^*, z^*, \alpha) &= \frac{1}{2}\alpha^T\alpha + \frac{1}{2\lambda}\alpha^T K\alpha - \alpha^T(K\alpha + y + \alpha) \\ &= -\frac{1}{2}\alpha^T\alpha - \frac{1}{2\lambda}\alpha^T K\alpha - \alpha^T y. \end{aligned}$$

The problem $\max_\alpha \mathcal{L}(w^*, z^*, \alpha)$ that we are left with is called the *dual problem*. We can solve it by again taking derivatives:

$$0 = \frac{\partial\mathcal{L}(w^*, z^*, \alpha)}{\partial\alpha} = -\alpha - \frac{1}{\lambda}K\alpha - y.$$

We finally end up with

$$\alpha^* = -\lambda(K + \lambda I)^{-1} y \tag{3.6a}$$

$$w^* = -\frac{1}{\lambda} \sum_{i=1}^{l} \alpha_i^* x_i. \tag{3.6b}$$

This is the linear ridge regression solution for the Lagrange multipliers $\alpha^* \in \mathbb{R}^l$ and the optimal hyperplane $w^* \in \mathbb{R}^d$. We can see that the penalty term has the effect of regularizing the inverse of K: The condition number of $(K + \lambda I)^{-1}$ is always bounded by λ, regardless of any small eigenvalues of K that might corrupt K^{-1}.

Kernelization. We note that the regression function ϕ^* corresponding to w^* can be written as

$$\phi^*(x) = (w^*)^T x = -\frac{1}{\lambda} \sum_{i=1}^{l} \alpha_i x_i^T x = y^T (K + \lambda I)^{-1} \beta$$

with $\beta = (x_1^T x, \ldots, x_l^T x)^T$. Since K and β are expressed solely with inner products, we can kernelize by replacing $x_i^T x_j \to k(x_i, x_j)$ and $x_i^T x \to k(x_i, x)$. This gives the *kernel ridge regression* formula:

$$\begin{aligned} \phi^*(x) &= y^T (K + \lambda I)^{-1} \beta, \\ K_{ij} &= k(x_i, x_j), \qquad i, j = 1, \ldots, l \\ \beta_i &= k(x_i, x), \qquad i = 1, \ldots, l \\ y^T &= (y_1, \ldots, y_l). \end{aligned} \tag{3.7}$$

Implicitly, (3.7) solves a linear ridge regression problem in feature space, i.e.

$$\min_{f \in \mathcal{H}} \left\{ \sum_{i=1}^{l} (y_i - \langle f, \Phi(x_i) \rangle)^2 + \lambda \|f\|_{\mathcal{H}}^2 \right\}.$$

3.2.2 Support vector machines

We go back to the binary classification problem, where $y_i = \pm 1$. The idea of support vector machines (SVMs) is to find a "maximum margin" hyperplane that separates the two classes. To this end, we introduce the hinge loss

$$r_h(x_i, y_i) = \max(0, 1 - y_i w^T x_i) \equiv (1 - y_i w^T x_i). \tag{3.8}$$

What does this loss function do? Suppose first that $y_i = +1$. Then $r_h(x_i, y_i) = 0$ if $w^T x_i \geq 1$, that is, if x_i is to the right of the blue shaded region in Figure 3.2. If x_i is

inside or to the left of the blue shaded region, then $r_h(x_i, y_i)$ increases linearly with the distance to the $\{x : w^T x = +1\}$ hyperplane[2]. The situation is reversed when $y_i = -1$: Now $r_h(x_i, y_i) = 0$ if x_i is to the left of the shaded region.

One can think of each hyperplane $\{x : w^T x = 0\}$ as being equipped with two companions $\{x : w^T x = +1\}$ and $\{x : w^T x = -1\}$. Together, they define a *margin region* (the blue region in Figure 3.2) where points are deemed "too close" to the hyperplane, thus the hinge loss is nonzero. Minimizing hinge loss tries to create a situation like the one depicted in Figure 3.2, with no datapoints inside the margin region. Note that the *margin* itself, i.e. the width of the margin region, is given by $2/\|w\|$. Rescaling w thus shrinks or inflates the margin region, without changing the hyperplane itself. So what should we do if we are looking for a hyperplane with *maximum margin*, that is, a margin region of maximal size such that no (or very few) data points are inside the margin region? Well, impose a penalty on $\|w\|$, of course.

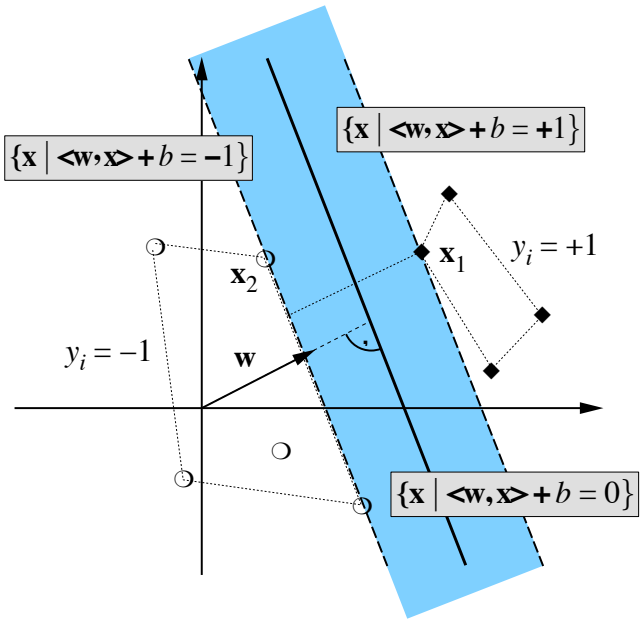

Figure 3.2 – Hinge loss explained: The hinge loss is always nonzero inside the blue shaded region, thus minimizing hinge loss tries to keep data points outside the region. The width of the region is $2/\|w\|$.

In other words, we pose the problem of finding a hyperplane $w \in \mathbb{R}^d$ that minimizes hinge loss with ridge regularization:

$$\min_{w \in \mathbb{R}^d} \left\{ \frac{1}{l} \sum_{i=1}^{l} r_h(x_i, y_i) + \lambda \|w\|^2 \right\}. \tag{3.9}$$

[2]We set $b = 0$ by the argument made in 2.1.

By introducing $c = 2/(\lambda l)$, we can rewrite this as

$$\min_{w \in \mathbb{R}^d} \left\{ c \sum_{i=1}^{l} (1 - y_i w^T x_i)_+ + \frac{1}{2} \|w\|^2 \right\} \tag{3.10}$$

which is suspiciously close to (3.4). With one difference however: Unlike the squared loss in (3.4), the hinge loss is not differentiable everywhere. This means that we cannot solve this problem directly by taking derivatives. Luckily we already know what to do: We apply the same strategy as before and turn the problem into a constrained optimization problem:

$$\min_{w \in \mathbb{R}^d, z \in \mathbb{R}^l} \left\{ c \sum_{i=1}^{l} z_i + \frac{1}{2} \|w\|^2 \right\}, \quad z_i \geq (1 - y_i w^T x_i),\ z_i \geq 0. \tag{3.11}$$

First, let's check that (3.11) is indeed equivalent to (3.10). For each z_i there are two constraints now. Both together enforce $z_i \geq (1 - y_i w^T x_i)_+$. Since the z_i are minimized, at the optimum we will have $z_i = (1 - y_i w^T x_i)_+$, which is indeed (3.10).

Now Lagrange duality states that (3.11) is equivalent to the saddle point problem

$$\max_{\alpha \geq 0, \mu \geq 0} \min_{z, w} \mathcal{L}(w, z, \alpha, \mu)$$

with the Lagrangian

$$\mathcal{L}(w, z, \alpha, \mu) = c \sum_{i=1}^{l} z_i + \frac{1}{2} \|w\|^2 + \sum_{i=1}^{l} \alpha_i (1 - y_i w^T x_i - z_i) + \sum_i \mu_i (-z_i)$$

where we have introduced Lagrange multipliers α and μ for the two types of constraints in (3.11), and since these are inequality constraints we maximize over $\alpha, \mu \geq 0$. Rearranging slightly gives

$$\mathcal{L}(w, z, \alpha, \mu) = \sum_{i=1}^{l} z_i (c - \mu_i - \alpha_i) + \frac{1}{2} \|w\|^2 + \sum_{i=1}^{l} \alpha_i - w^T \sum_{i=1}^{l} \alpha_i y_i x_i$$

Now we solve the inner problem:

$$\begin{aligned} 0 = \frac{\partial \mathcal{L}}{\partial z_i} &= c - \mu_i - \alpha_i && \Rightarrow \mu = c - \alpha, \\ 0 = \frac{\partial \mathcal{L}}{\partial w} &= w - \sum_{i=1}^{l} \alpha_i y_i x_i && \Rightarrow w^* = \sum_{i=1}^{l} \alpha_i y_i x_i. \end{aligned}$$

Hence

$$\mathcal{L}(w^*, z^*, \alpha, \mu) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j x_i^T x_j. \tag{3.12}$$

Note that μ has disappeared from (3.12), hence we don't have to optimize over it. But we still have $0 \leq \mu = c - \alpha$ by the first equation of the inner problem. This implies the *box constraint* $0 \leq \alpha \leq c$. In summary, we arrive at a quadratic optimization problem with linear inequality constraints, that is, a *quadratic program*:

$$\max_{0 \leq \alpha \leq c} \left\{ \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j x_i^T x_j \right\}. \tag{3.13a}$$

Quadratic programs are readily solved with many standard software packages - historically, SVMs have been a big factor in the development of quadratic programming in the 80ies and 90ies. Once we have solved for the α's, new points are classified according to

$$\operatorname{sgn}\left((w^*)^T x \right) = \operatorname{sgn}\left(\sum_{i=1}^{l} \alpha_i^* y_i x_i^T x \right). \tag{3.13b}$$

In the original optimization problem (3.10), we only considered hyperplanes $\{x : w^T x = 0\}$ passing through the origin. If one instead considers hyperplanes of the form $\{x : w^T x + b = 0\}$, then there is an additional linear constraint coming from the derivative of the Lagrangian with respect to b that needs to be incorporated in (3.13a):

$$\sum_{i=1}^{l} \alpha_i^* y_i = 0. \tag{3.13c}$$

Kernelization. Both the optimization problem (3.13a) and the classification rule (3.13b) can be kernelized. This leads to the *SVM optimization*

$$\max_{0 \leq \alpha \leq c} \left\{ \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \right\} \tag{3.14a}$$

and the *SVM classification rule*

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^{l} \alpha_i^* y_i k(x_i, x) \right). \tag{3.14b}$$

The additional constraint (3.13c), if present, is not changed under kernelization. An example with the gaussian kernel is illustrated in Figure 3.3.

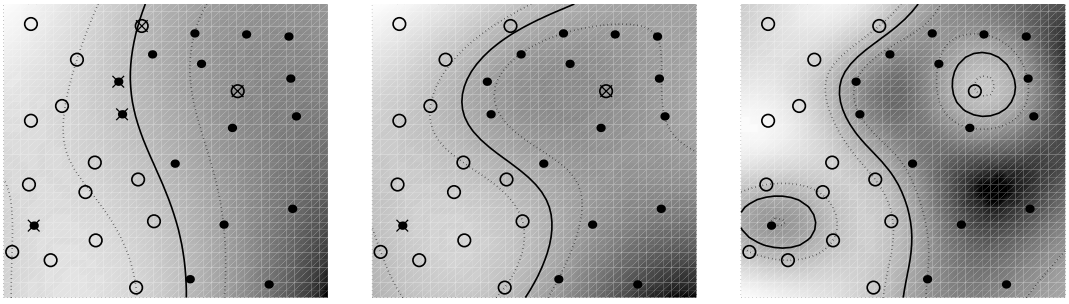

Figure 3.3 – A toy classification example with the gaussian kernel. The kernel width σ is decreased from left to right. Solid lines show the decision boundary, dotted lines the boundary of the margin region. As the kernel width decreases the margin region shrinks, the decision boundary becomes more complicated, and fewer training points are misclassified (Source: [SS98]).

Support vectors. An interesting observation is that the optimal parameter vector α^* is often sparse. We can distinguish two cases (see also Figure 3.4):

(i) $\alpha_i^* > 0$: The corresponding pair (x_i, y_i) "supports" the classification rule (3.14b) (hence the term support vector). As we shall see below, the vector x_i is inside or at the boundary of the margin region $\{x : |w^T x| \leq 1\}$.

(ii) $\alpha_i^* = 0$: The corresponding pair (x_i, y_i) does not support the classification rule (3.14b). First of all, this means that once the classifier is trained, we can forget about these "non-support vectors". Henceforth, we only need to evaluate the kernel on support vectors x_i paired with query points x, which is a big potential saving. But there is more: Loosely speaking, the corresponding constraint $z_i \geq 1 - y_i w^T x_i$ must hold without enforcing it in the optimization. This is so because the term that contains α_i in the Lagrangian $\mathcal{L}(w, z, \alpha, \mu)$ reads

$$\alpha_i(1 - y_i w^T x_i - z_i)$$

and we maximize over α_i. If the constraint $z_i^* \geq 1 - y_i w^T x_i$ was violated at the solution of the inner problem, then $\alpha_i(1 - y_i w^T x_i - z_i^*) > 0$ and maximizing over α_i would not result in $\alpha_i^* = 0$. Now recall that in the primal problem (3.11), we minimize z_i subject to the constraints $z_i \geq 1 - y_i w^T x_i$ and $z_i \geq 0$. If the former constraint holds for z_i^*, then minimizing z_i must give $z_i^* = 0$, which in turn implies $y_i w^T x_i \geq 1$. In other words, the non-support vector (x_i, y_i) is away from the margin region $\{x : |w^T x| \leq 1\}$: Non-support vectors are "well classified". What we see here is that not only do they not support the classification rule (3.14b), removing them from the optimization problem (3.14a) actually doesn't change the result.

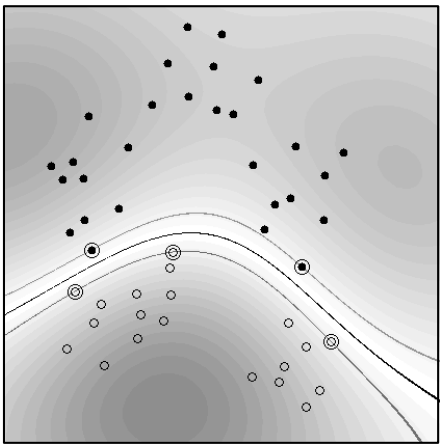

Figure 3.4 – Another toy classification example with the gaussian kernel. The decision boundary is denoted by the solid black line, the boundary of the margin region by the solid grey lines. Support vectors are circled. (Source: [SS98]).

Numerical solution. The task is now to solve (3.14a) numerically. We can distinguish two families of numerical methods for doing this. On the one hand, there are standard quadratic programming tools such as Quasi-Newton, conjugate gradient, and primal-dual interior point. All these methods work explicitly with the kernel matrix, which has to be stored in memory. For small datasets where storage is not an issue, these are usually the best methods. We refer the reader to standard textbooks on numerical optimization for details.

On the other hand, there are methods specialized to the specific form of the problem (3.14a). Among these, *sequential minimal optimization* (SMO) by Microsoft's John Platt stands out. The idea is to only update one pair α_i, α_j at a time (we need to do a pair due to the constraint (3.13c)). One can derive explicit update rules which makes this method pretty fast, and there is no need to keep the kernel matrix in memory. For details see [Pla98].

3.2.3 The VC dimension of support vector machines

In section 2.4.1 we discussed the VC dimension of the class of linear classifiers

$$f_w : \mathbb{R}^d \to \{\pm 1\}, \qquad f_w(x) = \operatorname{sgn}(w^T x + b)$$

as an example; the result was $V_{\mathcal{C}} = d + 1$. This is actually a corollary of the following result:

Theorem 3.4. m points in $\mathbb{R}^d$ can be shattered by hyperplanes if and only if one of the m points can be found such that if that point is used as the origin, the position vectors of the remaining $m-1$ points are linearly independent.

Proof. Exercise. Hint: Consider an arbitrary partition of the m points into sets S_1, S_2. Show that the convex hulls of S_1 and S_2 do not intersect if the linear independence assertion holds. Then $C_1 - C_2$ is also a convex set, and the origin is not in $C_1 - C_2$. Find a hyperplane that separates $C_1 - C_2$ from the origin. □

Kernel SVM's use hyperplanes in feature space as decision functions, consequently the VC dimension of kernel SVMs is $V_{\mathcal{C}} = d_{\mathcal{H}} + 1$ with $d_{\mathcal{H}}$ being the dimension of the feature space. That is bad news:

- SVMs with linear kernel: $V_{\mathcal{C}} = d + 1$
- SVMs with polynomial kernel of degree n: $V_{\mathcal{C}} = \binom{d+n}{n}$
- SVMs with gaussian kernel: $V_{\mathcal{C}} = \infty$.

Classifier	test error	reference
linear classifier	8.4%	[7]
3-nearest-neighbour	2.4%	[7]
SVM	1.4%	[8]
Tangent distance	1.1%	[45]
LeNet4	1.1%	[28]
Boosted LeNet4	0.7%	[28]
Translation invariant SVM	0.56%	[11]

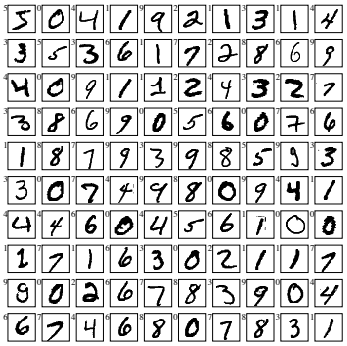

Figure 3.5 – Left: Performance comparison on the MNIST handwritten digits dataset. The SVM used a polynomial kernel of degree 9, corresponding to a feature space of dimension $\approx 3.2 \cdot 10^{20}$. Right: Sample from the MNIST dataset (Source: [SS98]).

So kernel SVMs have a huge or even infinite VC dimension! Our bounds of the form

$$\text{test error} \leq \text{training error} + \sqrt{\frac{V_{\mathcal{C}} \log l}{l}}$$

suggest that the generalization performance of SVMs should be terrible: We would have no way of deducing a small test error from a small training error. In practice though, that is not the case: SVMs can have an excellent generalization performance even though they operate in extremely high dimensional feature spaces, as the table 3.5 shows. So how can this be? Clearly there is something about the nature of the optimization problem

(3.10) that our VC bounds are not sensitive to: (3.10) prefers hyperplanes with a large margin. A key observation is now that maximum margin hyperplanes can have a small VC dimension! We will need the following definition:

Definition 3.5. Let $X_l = \{x_1, \ldots, x_l\} \subset \mathbb{R}^d$. The hyperplane $w^T x = 0$ is in *canonical form* with respect to X_l if $\min_{i=1,\ldots,l} |w^T x| = 1$.

A hyperplane is thus in canonical form if there are no points inside the margin region $\{x \in \mathbb{R}^d : |w^T x| \leq 1\}$, and at least one point is on the boundary of the margin region. Now we have the following

Theorem 3.6 (Vapnik 79')**.** Let $X_l = \{x_1, \ldots, x_l\} \subset \mathbb{R}^d$, and define the class of decision functions

$$\mathcal{C}_l = \left\{ f_w(x) = \operatorname{sgn}(w^T x) : w \text{ is in canonical form w.r.t. } X_l,\ \|w\| \leq \Lambda \right\}.$$

Then

$$V_{\mathcal{C}_l} \leq R^2 \Lambda^2,$$

where R is the radius of the smallest sphere around the origin containing X_l.

Proof. The proof is due to Gurvits (1997). Assume that the points $x_1, \ldots, x_r \in X_l$ can be shattered by the class $\mathcal{C}_l$, in another words, assume that $V_{\mathcal{C}_l} = r$. This means that for any subset of $x_1, \ldots, x_r$ there has to be a function $f_w \in \mathcal{C}_l$ that picks out that subset. In other words: For any $y_1, \ldots, y_r \in \{+1, -1\}$, there is $w \in \mathbb{R}^d$ such that $\|w\| \leq \Lambda$ and

$$y_i \langle w, x_i \rangle \geq 1, \quad i = 1, \ldots, r$$

since $f_w(x_i) = y_i$ for $i = 1, \ldots, r$ and w is in canonical form with respect to X_l. Summing these equations up gives

$$\left\langle w, \sum_{i=1}^{r} x_i y_i \right\rangle \geq r.$$

On the other hand, we can use the Cauchy-Schwarz inequality to get

$$\left| \left\langle w, \sum_{i=1}^{r} x_i y_i \right\rangle \right| \leq \|w\| \left\| \sum_{i=1}^{r} x_i y_i \right\| \leq \Lambda \left\| \sum_{i=1}^{r} x_i y_i \right\|.$$

Combining these two equations gives

$$r \leq \Lambda \left\| \sum_{i=1}^{r} x_i y_i \right\|. \tag{3.15}$$

Now comes the elegant part. Suppose that the labels $y_i \in \{-1, +1\}$ are chosen uniformly and independently at random: We simply toss a coin for each label y_i. Then

$$\mathbf{E}[\langle x_i y_i, x_j y_j \rangle] = \mathbf{E}[y_i y_j] \langle x_i, x_j \rangle = \delta_{ij} \langle x_i, x_j \rangle$$

and consequently

$$\mathbf{E}\left[\left\|\sum_{i=1}^{r} x_i y_i\right\|^2\right] = \mathbf{E}\left[\sum_{i,j=1}^{r} \langle x_i y_i, x_j y_j \rangle\right] = \sum_{i,j=1}^{r} \delta_{ij} \langle x_i, x_j \rangle = \sum_{i=1}^{r} \|x_i\|^2 \leq rR^2.$$

Now since the expectation value of $\| \sum_i x_i y_i \|$ is bounded by rR^2, there must be at least one realization $\tilde{y}_1, \ldots, \tilde{y}_r$ for which the bound actually holds:

$$\left\|\sum_{i=1}^{r} x_i \tilde{y}_i\right\|^2 \leq rR^2. \tag{3.16}$$

Combine (3.15) (which also holds for $\tilde{y}_1, \ldots, \tilde{y}_r$) with (3.16) to obtain $r \leq R^2 \Lambda^2$, which completes the proof. □

Recall that for SVMs the margin was given by $\delta = 1/\|w\|$. Therefore the bound in Theorem 3.6 implies

$$V_{\mathcal{C}_l} \leq R^2 / \delta_*^2 \tag{3.17}$$

where $\delta_* = \Lambda^{-1}$ is the smallest margin that functions in $\mathcal{C}_l$ are allowed to have. The intuitive picture is this: The smallest sphere enclosing the data defines a scale R, and the smallest allowed margin in $\mathcal{C}_l$ defines a resolution δ_* that functions in $\mathcal{C}_l$ can use to chop the sphere into pieces. The number of pieces we can chop the sphere into is a function of R/δ_*. The bound (3.17) gets smaller the smaller we can make Λ, but there is a limit to how small Λ can be: If it is too small, then $\mathcal{C}_l$ will be empty. This confirms that looking for maximum margin hyperplanes, as in (3.10), is a good idea.

Notice further that the bound (3.17) is independent of the dimension d, which saves kernel-SVMs. What is important is not the dimension of the feature space, but the resolution of the classifier compared to the resolution of the data.

There are two flaws in this argument. The first is that the SVM optimization (3.10) is not an optimization over the class $\mathcal{C}_l$: We do not impose bounds on $\|w\|$, and we do not require that no data points are inside the margin region, thus the function class in the optimization is much larger. Instead, the optimization problem penalizes large $\|w\|$ and points inside the margin region, so that suitable candidates for optimality "look like" functions in $\mathcal{C}_l$. Second, the function class $\mathcal{C}_l$ explicitly depends on the training data X_l. That is fundamentally different from the situation we always considered so far, where

the function class $\mathcal{C}$ was fixed, and only a particular member of the class was chosen based on the training data. For this reason, a rigorous theory of the generalization performance of SVMs is still an open issue nearly 40 years after their invention: There is no theory which guarantees that a given family of SVMs will have high accuracy on a given problem.

3.2.4 Nearest neighbour rules

We close this chapter with the discussion of some of the simplest classification rules and their kernelization: The *nearest neighbour* (NN) classifier and the *k nearest neighbor* (kNN) classifier. As usual we have training points $X_l = \{x_1, \dots, x_l\} \subset \mathbb{R}^d$ and labels $y_1, \dots, y_l$ which can have arbitrarily many classes. The NN-classifier is the function

$$f_l(x) = y_i \quad \Leftrightarrow \quad x_i \text{ is the nearest neighbor of } x \text{ in } X_l.$$

The function f_l is chosen based on the training data X_l. The corresponding function class is the smallest class $\mathcal{C}$ such that for any $l \in \mathbb{N}$ and any training data X_l, the function f_l is in the class $\mathcal{C}$. That is, $\mathcal{C}$ is the class of all Voronoi tesselations of $\mathbb{R}^d$ based on finitely many points. The NN-classifier thus has

- $V_{\mathcal{C}} = \infty$ (an arbitrary number of points can be shattered)
- zero empirical error (all training points are classified correctly).

This sounds bad. On the other hand, asymptotic performance of the NN-classifier is surprisingly good.

Theorem 3.7 (Cover & Hart '67)**.** Let $Y \in \{+1, -1\}$, and $L_l = \mathbf{P}[f_l(X) \neq Y]$. Then for any distribution of (X, Y), L_l converges in expectation:

$$\lim_{l \to \infty} \mathbf{E}[L_l] = \mathbf{E}[2\eta(X)(1 - \eta(X))] =: L_{NN}$$

with $\eta(x) = \mathbf{P}[Y = 1 | X = x]$.

Proof. In [DGL13], chapter 4. We just give the intuition behind the proof: Let X' be the nearest neighbor of X in X_l. Then $f_l(X) = Y'$, where the random variable Y' is independent of Y with $\mathbf{P}[Y' = 1] = \eta(X')$. Since X and X' will be close for large l, assume that $\mathbf{P}[Y' = 1] \approx \eta(X)$. Then

$$\begin{aligned} L_l = \mathbf{P}[Y' \neq Y] &= \mathbf{P}[Y' = 1, Y = 0] + \mathbf{P}[Y' = 0, Y = 1] \\ &= \mathbf{P}[Y' = 1]\mathbf{P}[Y = 0] + \mathbf{P}[Y' = 0]\mathbf{P}[Y = 1] \\ &= \eta(X)(1 - \eta(X)) + (1 - \eta(X))\eta(X). \end{aligned}$$

This is of course not a proof! □

As a corollary of Theorem 3.7, we have

$$L^* \leq L_{NN} \leq 2L^*(1 - L^*) \leq 2L^*.$$

Proof. Let $A = \min(\eta(X), 1 - \eta(X))$. Then

$$L^* \leq L_{NN} = 2\mathbf{E}[A(1 - A)] \leq 2\mathbf{E}[A]\mathbf{E}[1 - A] = 2L^*(1 - L^*) \leq 2L^*$$

since $L^* = \mathbf{E}[\min(\eta(X), 1 - \eta(X))]$, which follows from the proof of Theorem 2.6. □

Thus the NN-classifier has a very good asymptotic probability of error whenever L^* is small. If $L^* = 0$, then we even have $L_{NN} = 0$. In this case, the NN-classifier is weakly consistent (clf. section 2.2.2).

Since nearest neighbor relations are based on distances aka dot products, the NN-classifier can be kernelized to give the kernel-NN classifier:

$$f_l(x) = y_i \quad \leftrightarrow \quad \Phi(x_i) \text{ is nearest neighbor of } \Phi(x) \text{ in } X_l.$$

The reader is encouraged to write the corresponding algorithm in terms of kernels, without explicit use of the feature map Φ.

A generalization of the NN-classifier is the kNN-classifier, which considers the labels of the k nearest neighbors of x in X_l, the point x is then classified according to a majority vote among those labels. If there are only two classes and k is odd, then there are no problems with ties, otherwise one has to add a tie-breaking condition. For the corresponding probability of error one has, in the case of two classes and for all odd k,

$$L_{kNN} \leq L^* + \sqrt{\frac{2L_{NN}}{k}}.$$

For $L^* = 0$, the kNN-classifier is therefore also weakly consistent. Perhaps more interestingly, $\lim_{k\to\infty} L_{kNN} = L^*$ even if $L^* > 0$. This suggests that the kNN-classifier is consistent if we increase k as we increase l, and this is indeed the case. The kNN-classifier can of course also be kernelized.

4 Unsupervised Learning

In this chapter we will look at unsupervised learning. Unlike in the supervised case that we looked at so far, we only have data points $\{x_1, \ldots, x_l\} = X_l \subset \mathbb{R}^d$ this time, and no labels. This is a fundamentally different setup! In the supervised case, our task was to learn the conditional probability distribution of Y given $X = x$. But here there is no variable Y - we just don't have a well defined learning problem to start with. So before going ahead, we have to pose the problem by telling ourselves what it is that we are looking for. This is why there is a huge zoo of methods in unsupervised learning, and many of them are not easily compared because they make different assumptions and look for different things in the data.

One often assumes that the x_i are i.i.d. realizations of a random variable X (even if that assumption may be difficult to verify in practice). We can thus either try to learn something about the distribution of X, assuming that $x_i \sim X$ holds, or we can just try to find structure in the data $\{x_1, \ldots, x_l\}$ itself.

4.1 Two typical tasks and two ad hoc methods

The two most common tasks in unsupervised learning are *clustering* and *dimensionality reduction*.

4.1.1 Clustering

In clustering, one looks for groups in the data $X_l = \{x_1, \ldots, x_l\}$. Given a metric $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ which we can think of as a "similarity measure" between data points, the task is to find clusters $S_1, \ldots, S_k \subset X_l$ such that the similarity of points within clusters is larger then the similarity of points in different clusters.

Ad hoc method: k-means. For a given and fixed k, this methods tries to partition X_l into k groups $S_1, \ldots, S_k$ (that is, such that $X_l = S_1 \cup \ldots \cup S_k$ and $S_i \cap S_j = \emptyset$) by minimizing the "within cluster sum of squares". That is, the objective is to find

$$\operatorname*{argmin}_{\mathbf{S},\mu} C(\mathbf{S}, \mu) = \operatorname*{argmin}_{\mathbf{S},\mu} \sum_{i=1}^{k} \sum_{x \in S_i} d(x, \mu_i)^2$$

where μ_i is the center of cluster S_i (which is also subject to minimization). This is a nonconvex optimization problem, and finding the optimal solution is NP hard for general d (even if $k = 2$) or general k (even if $d = 2$). Nevertheless this method is extremely popular due to a cheap iterative refinement technique that is guaranteed to converge to a local optimum. This technique, typically called the *k-means algorithm*, consists of iterating the following two steps after initial center locations $\mu_1^{(0)}, \ldots, \mu_k^{(0)}$ have been given:

(i) assignment step: we keep $\mu^{(n)}$ fixed and minimize $C(\mathbf{S}, \mu^{(n)})$ with respect to $\mathbf{S}$. That is, we assign data points to groups $S_1^{(n)}, \ldots, S_k^{(n)}$ by nearest neighbor classification, i.e. $S_j^{(n)}$ consists of all points $x \in X_l$ that are closer to $\mu_i^{(n)}$ then to the other centers $\mu_j^{(n)}$.

(ii) update step: We keep $\mathbf{S}^{(n)}$ fixed and minimize $C(\mathbf{S}^{(n)}, \mu)$ with respect to μ. That is, we calculate new center positions

$$\mu_i^{(n+1)} = \frac{1}{|S_i^{(n)}|} \sum_{x \in S_i^{(n)}} x.$$

This algorithm is cheap since only kl distances need to be computed in each iteration (and typically $k \ll l$), and it converges quickly to a local optimum. In order to find a good global solution, one typically runs the algorithm several times with different randomly drawn initial locations $\mu_1^{(0)}, \ldots, \mu_k^{(0)}$. The algorithm is also ad hoc in the sense that results strongly depend on the metric d, the number k of clusters has to be chosen *a priori*, and it has a bias towards equally sized clusters.

Dimensionality reduction

Here, the task is to find a feature map $\Phi : \mathbb{R}^d \to F$, but this time we require $s = \dim F \ll d$, that is we look for a mapping to a *low-dimensional feature space*. Additionally, this map should preserve some structure that has to be specified (and different methods will make different assumptions as to what that structure is).

Ad hoc method: PCA. Principal component analysis (PCA) is one of the most common *linear* dimensionality reduction techniques. The task is to find a linear map Φ such that the "percentage of variance explained" by Φ is maximized. We'll get to what that means later. PCA first arranges the data in a matrix (that we also call X with some abuse of notation)

$$X = \begin{pmatrix} - & x_1 & - \\ - & x_2 & - \\ & \cdots & \\ - & x_l & - \end{pmatrix} \in \mathbb{R}^{l \times d}.$$

We assume that X is mean-free, that is, $\sum_{i=1}^{l} x_i = 0$. Otherwise the mean is subtracted from each row. Then one performs a singular value decomposition (SVD) of X:

$$X = U\Sigma V^T, \quad U \in \mathbb{R}^{l \times l}, \Sigma \in \mathbb{R}^{l \times d}, V \in \mathbb{R}^{d \times d}.$$

The matrix Σ is the matrix of singular values, i.e. the upper $d \times d$ block of Σ is equal to $\mathrm{diag}(\sigma_1, \ldots, \sigma_d)$ with the (real) singular values $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_d$, and the lower block of the matrix is zero. The matrix

$$V = \begin{pmatrix} | & | & & | \\ v_1 & v_2 & \ldots & v_d \\ | & | & & | \end{pmatrix}$$

is the matrix of singular vectors $v_i \in \mathbb{R}^d$. PCA consists of keeping the singular vectors corresponding to the largest s singular values, these singular vectors are then called the *principal components*. The associated feature map is

$$\Phi(x_i) = \left(x_i^T v_1, \ldots, x_i^T v_s \right).$$

The principal components are mutually orthogonal and maximize explained variance in the sense that

$$v_1 = \operatorname*{argmax}_w \frac{w^T X^T X w}{w^T w},$$

so v_1 is the direction of largest variance (the matrix $X^T X$ is the covariance matrix), v_2 is the direction of largest variance in the subspace orthogonal to v_1, and so on. PCA is cheap and works well when linear projections make sense, but doesn't work well when they don't. See figure 4.1.

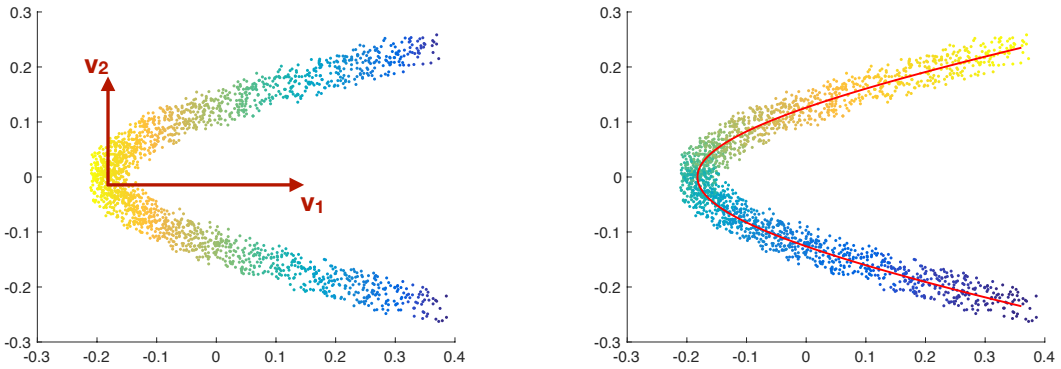

Figure 4.1 – Left: PCA on a toy dataset. Arrows show the principal components v_1 and v_2. v_1 is in the direction of largest variance. The color of the points shows the first component $x^T v_1$ of the feature map. Right: The underlying parabola is shown in red. Points are colored according to a feature map that parameterizes $\mathcal{M}$.

4.2 Manifold learning

Manifold learning is a subset of dimensionality reduction methods. One makes the following *standing assumption*:

Let $x_1, \ldots, x_l$ be i.i.d. realizations of a random variable X, which is distributed according to a density q with respect to the Lebesgue measure in $\mathbb{R}^d$ (we write $X \sim q$ as shorthand). Let $\text{supp}(q) = \mathcal{M} \subset \mathbb{R}^d$ where $\mathcal{M}$ is a compact submanifold of $\mathbb{R}^d$.

It is important to realize that we do not know the distribution q nor the manifold $\mathcal{M}$; all we see is the data $\{x_1, \ldots, x_l\}$ and the manifold structure is "hidden" in it. Our task is to find a map $\Phi : \mathbb{R}^d \to F$ such that

- Φ parameterizes $\mathcal{M}$
- $\|\Phi(x) - \Phi(y)\|_F$ is an intrinsic distance in $\mathcal{M}$.

The example in Figure 4.1 shows that PCA does not work here. In this example, $\mathcal{M}$ is a "thick" hyperbola. We would expect a one-dimensional feature map $\Phi : \mathbb{R}^2 \to \mathbb{R}$ that parameterizes the underlying hyperbola. Such a Φ would monotonically decrease from the top right corner to the center left and further to the bottom right corner (see Figure 4.1 on the right). Instead, the first PCA component does not differentiate the two branches of the hyperbola and consequently groups points in the top right and bottom right corner, which are far away on $\mathcal{M}$, close together.

A somewhat more interesting example is shown in Figure 4.2. Here the points x_i are images, which live in a high-dimensional vector space. But the images have a "hidden"

structure: They all show the same face but from different angles, which corresponds to a two-dimensional manifold in the (much larger) space of all possible pictures. Manifold learning finds a map Φ into a two-dimensional feature space where the features are the left-right and top-down angle, thus the two-dimensional manifold is successfully uncovered and parameterized.

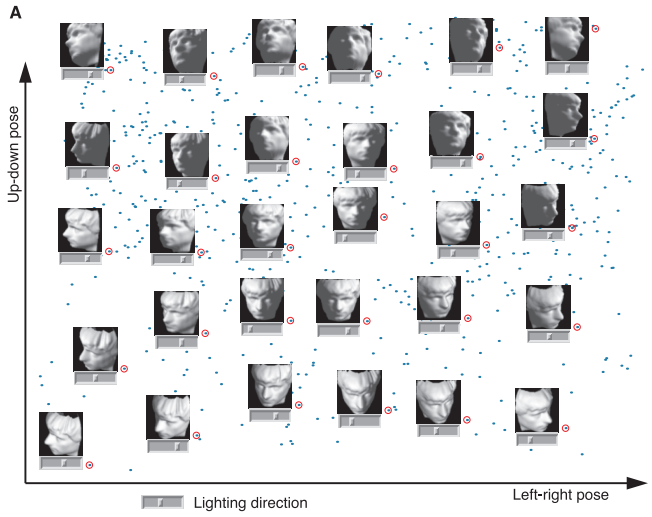

Figure 4.2 – A two-dimensional embedding of images showing the same face at different angles produced by the Isomap algorithm. From [TDSL00].

4.2.1 First attempt: Multidimensional scaling

Multidimensional scaling (MDS) goes back to Kruskal and the 1960ies [Kru64]. Given a metric d on $\mathbb{R}^d$, one tries to find a $\Phi : \mathbb{R}^d \to F$ with $\dim F = s \ll d$ such that $d(x_i, x_j) \approx \|\Phi(x_i) - \Phi(x_j)\|_F$. This is done by minimizing the *stress*

$$S[\Phi] = \sum_{ij} (d_{ij} - \|\Phi(x_i) - \Phi(x_j)\|_F)^2, \qquad d_{ij} = d(x_i, x_j).$$

This algorithm doesn't actually need the points $\{x_1, \ldots, x_l\}$ themselves, only their pairwise distances. Indeed this was historically how the algorithm was first used. The typical scenario is "given a list of flight distances between cities, what are the coordinates of the cities?". However, for us MDS has a huge problem: The distance d is not an intrinsic distance on $\mathcal{M}$! It is in fact an *extrinsic* distance; a distance in the ambient space $\mathbb{R}^d$. As such, it is meaningless in general, see for example Figure 4.3 (A). Since MDS tries to reconstruct all pairwise distances d_{ij}, it implicitly "trusts" all

these distances as well, which we cannot do. Locally, however, things are fine: There is a scale $\varepsilon > 0$ such that d_{ij} is approximately an intrinsic distance whenever $d_{ij} \leq \varepsilon$. This is because locally $\mathcal{M}$ is flat and is thus well approximated by a tangent plane, and distances between points in the tangent plane can be measured by d. Of course we have no knowledge of the scale ε at which $\mathcal{M}$ looks flat; estimating this ε is one of the major open issues in manifold learning.

We have established that we can only trust distances measured by d between points that are close. Manifold learning algorithms thus typically do the following:

1. construct a neighborhood graph G from the data X_l and the metric d. This is typically either a kNN graph (each point in X_l is connected with its k nearest neighbors) or a εNN graph (points such that $d(x_i, x_j) \leq \varepsilon$ are connected).
2. construct an intrinsic distance d_G using the graph G, e.g. the shortest path distance.
3. find an embedding $\Phi : \mathbb{R}^d \to F$ such that $d_G(x_i, x_j) \approx \|\Phi(x_i) - \Phi(x_j)\|_F$.

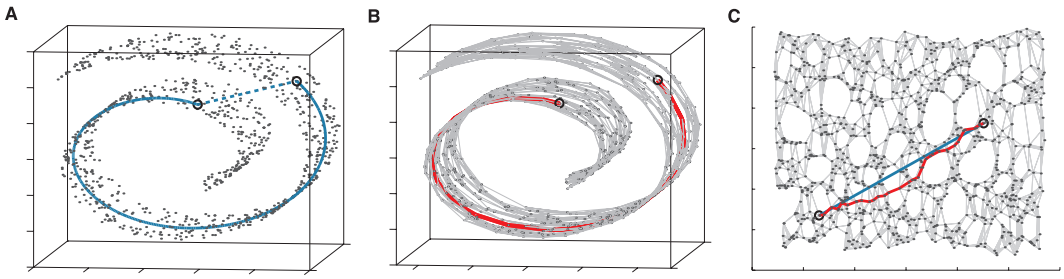

Figure 4.3 – (A) points sampled from the swiss roll, a two-dimensional submanifold in $\mathbb{R}^3$. The extrinsic distance between points (dotted line) can be massively different from a meaningful intrinsic distance (e.g. geodesic, solid line). (B) The neighborhood graph G, and the shortest in G (red line). (C) neighborhood graph, geodesic and shortest path under the mapping Φ. From [TDSL00].

In the following we will discuss two very popular manifold learning algorithms that both follow this layout - the difference is how the steps (1) - (3) are performed.

4.2.2 Isomap

Isomap [TDSL00] performs the following steps:

1. G is either the kNN graph or the εNN graph (for fixed $k \in \mathbb{N}$ and $\varepsilon > 0$ respectively). Both variants exist. Edges in G either have weights $w_{ij} = 1$ (unweighted) or $w_{ij} = d(x_i, x_j)$ (weighted by extrinsic metric d).

2. d_G is the shortest path distance in G.

3. Φ is found by minimizing the stress

$$S[\Phi] = \sum_{i,j} \left(d_G(x_i, x_j) - \|\Phi(x_i) - \Phi(x_j)\|_F\right)^2, \tag{4.1}$$

in other words MDS is applied to the matrix d_G of pairwise shortest path distances.

The idea here is that the shortest path distance d_G approximates the geodesic distance in $\mathcal{M}$. Since geodesic distance is a meaningful intrinsic distance, this allows us to use MDS to find an embedding Φ of the manifold $\mathcal{M}$. The fact that the shortest path distance in G has something to do with the geodesic distance in $\mathcal{M}$ is intuitively rather obvious, but surprisingly it took 12 years to really clear that up, and that clearing up came with a surprising insight. We will look at this in a moment, but let's actually define what geodesic distance is first.

Definition 4.1 (geodesic distance)**.** Let $\mathcal{M} \subset \mathbb{R}^d$ be a submanifold, let $f : \mathcal{M} \to \mathbb{R}^+$ be continuous, and $x, y \in \mathcal{M}$. Let $\Gamma_{xy} = \{\gamma : [0,1] \to \mathcal{M} : \gamma(0) = x, \gamma(1) = y, \gamma \in C^1\}$ be the set of all paths in $\mathcal{M}$ connecting x and y. We call

$$D_f(x, y) = \min_{\gamma \in \Gamma_{xy}} \int_0^1 f(\gamma(t))|\gamma'(t)|dt$$

the *f-geodesic distance* (one can check that this is really a distance). $D_1 \equiv D$ is called the *geodesic distance*.

To appreciate the theorem that follows, recall that G is a *random graph* since it is constructed from the randomly drawn data $\{x_1, \ldots, x_l\}$. The shortest path distance in G is thus a random variable, and convergence will be in probability.

Theorem 4.2 (Alangir, v.Luxburg 2012)**.** Let $G_{k,l}$ be the kNN graph based on i.i.d. samples $x_1, \ldots, x_l$ drawn from the density q with $\text{supp}(q) = \mathcal{M}$. Let $d_{G_{k,l}}$ be the shortest path distance in $G_{k,l}$. As $k \to \infty$, $l \to \infty$, and $k/l \to 0$:

(a) If $G_{k,l}$ has edge weights $w_{ij} = d(x_i, x_j)$, then

$$d_{G_{k,l}}(x_i, x_j) \to D(x_i, x_j).$$

(b) If $G_{k,l}$ has edge weights $w_{ij} = 1$, then

$$d_{G_{k,l}}(x_i, x_j) \to D(x_i, x_j), \qquad f(x) = q(x)^{1/d}.$$

In both cases, convergence is in probability.

Proof. Can be found in [AVL12]. □

Some remarks are in order.

- As Theorem 4.2 shows, Isomap finds an embedding Φ of the submanifold $\mathcal{M}$ according to geodesic distance in $\mathcal{M}$ if $w_{ij} = d(x_i, x_j)$, that is, if edges in G are weighted according to their *extrinsic distance*. Recall that edges in G represent exactly those extrinsic distances we decided are sufficiently close to intrinsic distances and thus trustworthy.
- If one uses $w_{ij} = 1$, i.e. an unweighted kNN graph, then d_G converges to a weighted geodesic distance which assigns a higher cost for crossing regions with higher density q. This means that the Isomap distance in this situation will try to avoid high density regions, which is usually bad. We can understand why this happens: In a kNN graph, edges in a region with low sampling density will usually be longer. If crossing any edge has cost $w_{ij} = 1$, then it is cheaper to cross low-density regions. This doesn't happen in εNN graphs since edges have at most length ε.
- The solution $\Phi : \mathbb{R}^d \to F$ to the minimization problem 4.1 is found by diagonalising the matrix HSH, where S is the matrix of squared shortest path distances $S_{ij} = d_G^2(x_i, x_j)$ and H is the centering matrix $H_{ij} = \delta_{ij} - 1/l$. One picks the dimension $s = \dim F$ of the embedding space, Φ is then given by the eigenvectors of the s largest eigenvalues of HSH.

A Appendix

A.1 Measure Theory

A.1.1 Basic definitions and properties

We collect the most basic definitions in measure theory, followed by some results which will be useful in the lectures.

Definition A.1 (Algebra and σ-algebra). Consider a collection $\mathcal{A}$ of subsets of a set X, and the following properties:

(a) When $A \in \mathcal{A}$ then $A^c := X \setminus A \in \mathcal{A}$.

(b) When $A, B \in \mathcal{A}$ then $A \cup B \in \mathcal{A}$.

(b') Given a finite or infinite countable sequence $\{A_k\}$ of subsets of X, $A_k \in \mathcal{A}$, then also $\bigcup_k A_k \in \mathcal{A}$.

If $\mathcal{A}$ satisfies (a) and (b), it is called an *algebra* of subsets of X; if it satisfies (a) and (b'), it is called a *σ-algebra* .

It follows from the definition that a σ-algebra is an algebra, and for an algebra $\mathcal{A}$ holds

- $\emptyset, X \in \mathcal{A}$;
- $A, B \in \mathcal{A} \Rightarrow A \cap B \in \mathcal{A}$;
- $A, B \in \mathcal{A} \Rightarrow A \setminus B \in \mathcal{A}$;
- if $\mathcal{A}$ is a σ-algebra , then $\{A_k\} \subset \mathcal{A} \Rightarrow \bigcap_k A_k \in \mathcal{A}$.

Definition A.2 (Measure). A function $\mu : \mathcal{A} \to [0, \infty]$ on a σ-algebra $\mathcal{A}$ is a *measure* if

(a) $\mu(\emptyset) = 0$;

(b) $\mu(A) \geq 0$ for all $A \in \mathcal{A}$; and

(c) $\mu\left(\bigcup_k A_k\right) = \sum_k \mu(A_k)$ if $\{A_k\}$ is a finite or infinite sequence of pairwise disjoint sets from $\mathcal{A}$, that is, $A_i \cap A_j = \emptyset$ for $i \neq j$. This property of μ is called *σ-additivity* (or *countable additivity*).

If, in addition, $\mu(X) = 1$, then μ is called a *probability measure*.

Definition A.3.

(a) If $\mathcal{A}$ is a σ-algebra of subsets of X and μ is a measure on $\mathcal{A}$, then the triple $(X, \mathcal{A}, \mu)$ is called a *measure space*. The subsets of X contained in $\mathcal{A}$ are called *measurable*. The pair $(X, \mathcal{A})$ is called *a measurable space*.

(b) If $\mu(X) = 1$, then the measure space is called a *probability space*.

A.1.2 Lebesgue integration

Definition A.4 (Borel σ-algebra / measure)**.** Let X be a topological space. The smallest σ-algebra containing all open subsets of X is called the *Borel σ-algebra* . If $\mathcal{A}$ is the Borel σ-algebra , then a measure μ on $\mathcal{A}$ is a *Borel measure* if the measure of any compact set is finite.

For example, if $X = \mathbb{R}^d$, then the Borel σ-algebra is the smallest algebra containing all rectangles of the form $[x_1, y_1) \times \ldots \times [x_d, y_d)$. One example of a Borel measure on $\mathbb{R}^d$ is the *Lebesgue measure*, which assigns to each rectangle in $\mathbb{R}^d$ its volume:

$$\mu\left([x_1, y_1) \times \ldots \times [x_d, y_d)\right) = |y_1 - x_1| \times \ldots \times |y_d - x_d|.$$

Definition A.5 (Measurable function)**.** Let $(X, \mathcal{A}, \mu)$ and $(Y, \mathcal{B}, \nu)$ be measure spaces, and $f : X \to Y$ a function. We call f *measurable*, if $f^{-1}(B) \in \mathcal{A}$ whenever $B \in \mathcal{B}$.

Lebesgue integration is concerned with integrals of measurable functions where $Y = \mathbb{R}$ (or $\mathbb{C}$) and $\mathcal{B}$ is the Borel σ-algebra on $\mathbb{R}$. For a detailed construction of the Lebesgue integral we refer to any textbook on measure theory.

Definition A.6 (Lebesgue integral)**.** Let f_n be *simple function* of the form

$$f_n = \sum_{i=1}^{n} \lambda_i I_{A_i}, \qquad I_A(x) = \begin{cases} 1, & x \in A, \\ 0, & x \notin A. \end{cases}$$

where $\lambda_i \in \mathbb{R}$ and the A_i are disjoint measurable sets. The Lebesgue integral of f_n is given by

$$\int f_n \, d\mu := \sum_{i=1}^{n} \lambda_i \mu(A_i),$$

A measurable function f is called (Lebesgue) integrable if for any convergent approximations of it by simple functions f_n the limit $\lim_{n\to\infty} \int f_n d\mu$ exists and is unique.

A.1.3 Useful inequalities and convergence theorems

Note: This section is here mostly for completeness and reference purposes.

Definition A.7 (L^p space)**.** For $p \in (0, \infty)$ the space $L^p_\mu(X)$ (sometimes also denoted as $L^p(X, \mu)$) consists of the equivalence classes[1] of measurable functions $f : X \to \mathbb{C}$ such that $\int |f|^p \, d\mu < \infty$. For $p \geq 1$, the L^p norm is defined by $\|f\|_p = (\int |f|^p d\mu)^{1/p}$. The space $L^\infty_\mu(X)$ consists of equivalence classes of essentially bounded functions.

If μ is finite, then $L^\infty_\mu(X) \subset L^p_\mu(X)$ for every $p > 0$.

Hölder's inequality gives another connection between functions in L^p spaces: if $p \in [1, \infty]$ and q are such that $1/p + 1/q = 1$ (with the convention $1/\infty = 0$), $f \in L^p_\mu(X)$, and $g \in L^q_\mu(X)$, then one has

$$\int |fg| \, d\mu = \|fg\|_1 \leq \|f\|_p \|g\|_q.$$

For $p = 2$ the norm $\|\cdot\|_2$ comes from the inner product

$$\langle f, g \rangle = \int fg \, d\mu,$$

therefore $L^2_\mu(X)$ is a Hilbert space.

For sequences of functions we have the following results concerning interchangeability of integration and limits.

Theorem A.8 (Fatou's lemma)**.** For a sequence $\{f_n\}$ of non-negative measurable functions, define $f : X \to [0, \infty]$ as the a.e. pointwise limit

$$f(x) = \liminf_{n\to\infty} f_n(x).$$

Then, f is measurable and

$$\int f \, d\mu \leq \liminf_{n\to\infty} \int f_n \, d\mu \, .$$

[1] Two measurable functions are *equivalent* if they coincide up to a set of measure zero.

Theorem A.9 (Lebesgue dominated convergence theorem)**.** Let $f : X \to [-\infty, \infty]$, $g : X \to [0, \infty]$ be measurable functions, and $f_n : X \to [-\infty, \infty]$ be measurable functions such that $|f_n(x)| \leq g(x)$ and $f_n(x) \to f(x)$ as $n \to \infty$ a.e. If g is integrable, then so are f and the f_n, furthermore

$$\lim_{n\to\infty} \int f_n \, d\mu = \int f \, d\mu \,.$$

Note that non-negative integrable functions define finite measures: Let $f : X \to [0, \infty]$ be integrable, then $\mu_f : \mathcal{A} \to [0, \infty]$, defined via

$$\mu_f(A) := \int_A f \, d\mu = \int f I_A \, d\mu$$

is a measure. Here is the converse result:

Theorem A.10 (Radon–Nikodym)**.** Let $(X, \mathcal{A}, \mu)$ be a finite measure space, and $\nu : \mathcal{A} \to [0, \infty)$ a second measure with the property[2] $\nu(A) = 0$ whenever $\mu(A) = 0$. Then there exists a non-negative integrable function $f : X \to [0, \infty]$ such that

$$\nu(A) = \int_A f \, d\mu \,.$$

Theorem A.11 (Fubini)**.** Let $(X, \mathcal{A}, \mu)$ be the product of the measure spaces $(X_i, \mathcal{A}_i, \mu_i)$, $i = 1, 2$, and let a μ-integrable function $f : X \to \mathbb{R}$ be given. Then, for a.e. $x_1 \in X_1$ the function $x_2 \mapsto f(x_1, x_2)$ is μ_2-integrable. Furthermore, the function

$$x_1 \mapsto \int_{X_2} f(x_1, x_2) \, d\mu_2(x_2)$$

is μ_1-integrable, and

$$\int_{X_1} \left(\int_{X_2} f(x_1, x_2) \, d\mu_2(x_2) \right) d\mu_1(x_1) = \iint_X f(x_1, x_2) \, d\mu(x_1, x_2) \,.$$

A.2 Probability Theory

A.2.1 Probability Spaces, Random Variables

Definition A.12 (Probability space)**.** A measure space $(\Omega, \mathcal{F}, \mathbf{P})$ is called a *probability space* if $\mathbf{P}(\Omega) = 1$. In this case, Ω is called the *sample space*, $\mathcal{F}$ is called *event space* and $\mathbf{P}$ is called a *probability measure*.

Note that the switch in notation from $(X, \mathcal{A}, \mu)$ to $(\Omega, \mathcal{F}, \mathbf{P})$ is intended: These are the

[2] We say ν is *absolutely continuous* with respect to μ, in shorthand $\nu \ll \mu$, iff $(\mu(A) = 0) \Rightarrow (\nu(A) = 0)$.

customary notations in the context of measure theory resp. probability theory.

Definition A.13 (Random Variable)**.** Let $(\Omega, \mathcal{F}, \mathbf{P})$ be a probability space and $(Y, \mathcal{B})$ a measurable space. A *random variable* is a measurable function (in the sense of definition 5) $X : \Omega \to Y$. The random variable X induces a probability measure μ on $(Y, \mathcal{B})$ by

$$\mu(B) = \mathbf{P}[\{\omega : X(\omega) \in B\}] \equiv \mathbf{P}[X \in B]$$

The probability measure μ is called the *distribution* of the random variable X.

Typically (and from now on), $Y = \mathbb{R}$ and $\mathcal{B}$ is the Borel σ-algebra on $\mathbb{R}$. In practice, the probability space $(\Omega, \mathcal{F}, \mathbf{P})$ is treated as a technical device only. One usually works with the distribution μ of X directly in order to carry out all computations involving X.

Definition A.14 (Expectation value)**.** Let $X : \Omega \to \mathbb{R}$ be a random variable. The *expectation value* of X is the integral:

$$\mathbf{E}[X] = \int_\Omega X(\omega) d\mathbf{P}(\omega) = \int_{\mathbb{R}} x\mu(dx),$$

if it exists.

Definition A.15 (Variance)**.** Let $X : \Omega \to \mathbb{R}$ be a random variable. The *variance* of X is

$$\mathbf{Var}[X] = \mathbf{E}\left[(X - \mathbf{E}[X])^2\right]$$

if $\mathbf{E}[X] < \infty$. If $\mathbf{E}[X] = \infty$ then $\mathbf{Var}[X] = \infty$.

Definition A.16 (Independence)**.** Let $X_1, \ldots, X_n$ be random variables. They induce the measure $\mu^{(n)}$ on the Borel σ-algebra of $\mathbb{R}^n$ with the property

$$\mu^{(n)}(B_1 \times \ldots \times B_n) = \mathbf{P}[X_1 \in B_1, \ldots, X_n \in B_n], \quad B_1, \ldots, B_n \in \mathcal{B}.$$

$\mu^{(n)}$ is called the *joint distribution* of the random variables $X_1, \ldots, X_n$. If $\mu^{(n)}$ is the product measure, that is, if

$$\mu^{(n)}(B_1 \times \ldots \times B_n) = \mathbf{P}[X_1 \in B_1] \times \ldots \times \mathbf{P}[X_n \in B_n], \quad B_1, \ldots, B_n \in \mathcal{B},$$

then the random variables $X_1, \ldots, X_n$ are called *independent*. The events $A_1, \ldots, A_n \in \mathcal{F}$ are independent if the random variables $I_{A_1}, \ldots, I_{A_n}$ are independent.

Fubini's theorem (theorem 11) implies the following:

Theorem A.17. If the random variables $X_1, \ldots, X_n$ are independent and have finite expectations then

$$\mathbf{E}[X_1 X_2 \ldots X_n] = \mathbf{E}[X_1]\mathbf{E}[X_2] \ldots \mathbf{E}[X_n].$$

A.2.2 Conditional Expectation and Conditional Probabilities

We will mostly be ok with the following version of conditional expectation, which holds for events with finite probability.

Definition A.18 (Conditional Expectation)**.** If $X : \Omega \to \mathbb{R}$ is a random variable and $A \in \mathcal{F}$ is an event with positive probability $\mathbf{P}[A] > 0$, then the *conditional expectation* of X given A is defined by

$$\mathbf{E}[X|A] = \frac{\mathbf{E}[XI_A]}{\mathbf{P}[A]}.$$

The *conditional probability* of an event $B \in \mathcal{F}$ given A is

$$\mathbf{P}[B|A] = \mathbf{E}[I_B|A] = \frac{P[A \cap B]}{P[A]}.$$

Sometimes, one wishes to condition on events A with $\mathbf{P}[A] = 0$ - that is typically the case when working with continuous random variables. For completeness, here is the definition of conditional expectation that holds in such cases, and is even much more general - we can condition a random variable Y *on another random variable* X. It takes a while to digest it!

Definition A.19 (Conditional Expectation II)**.** Let $Y : \Omega \to \mathbb{R}$ be a random variable with finite expectation and $X : \Omega \to \mathbb{R}^d$ be another random variable. Let $\mathcal{F}_X$ be the σ-algebra generated by X:

$$\mathcal{F}_X = \left\{ X^{-1}(B); B \in \mathcal{B}^n \right\}.$$

The *conditional expectation* $\mathbf{E}[Y|X]$ of Y given X is a random variable with the property that for all $A \in \mathcal{F}_X$

$$\int_A Y d\mathbf{P} = \int_A E[Y|X] d\mathbf{P}.$$

Bibliography

[AVL12] Morteza Alamgir and Ulrike Von Luxburg. Shortest path distance in random k-nearest neighbor graphs. *arXiv preprint arXiv:1206.6381*, 2012. 50

[BBL04] Olivier Bousquet, Stéphane Boucheron, and Gábor Lugosi. Introduction to statistical learning theory. In *Advanced lectures on machine learning*, pages 169–207. Springer, 2004. 5

[DGL13] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013. 5, 9, 19, 22, 40

[Kru64] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964. 47

[Mit97] Tom M Mitchell. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37):870–877, 1997. 1

[Pla98] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998. 36

[Roj13] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013. 5

[Sam00] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 44(1.2):206–226, Jan 2000. 1

[SS98] Alex J Smola and Bernhard Schölkopf. *Learning with kernels*. GMD-Forschungszentrum Informationstechnik, 1998. 25, 35, 36, 37

[TDSL00] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000. 47, 48

[Tur50] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. 1

[VV98] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998. 2, 5, 9, 20