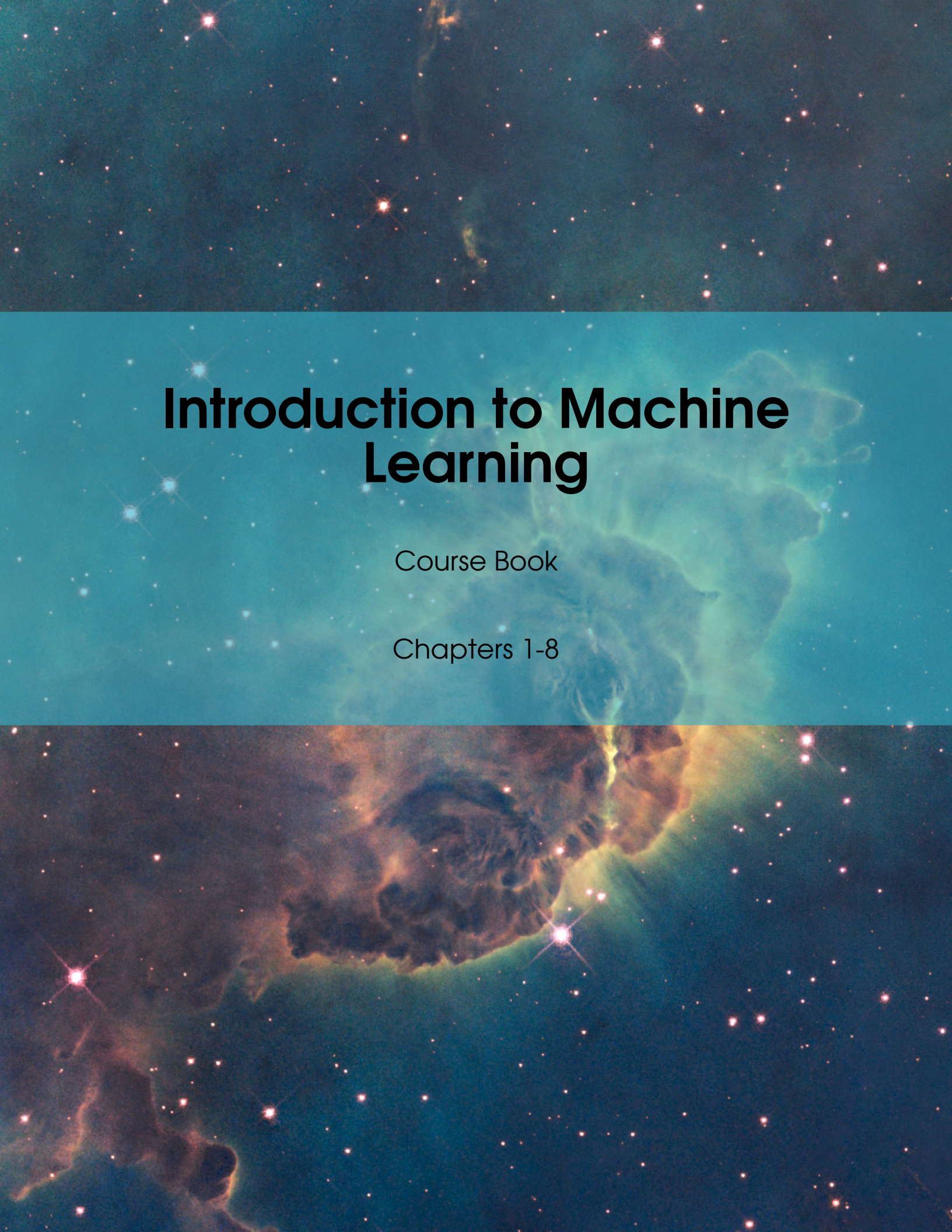


Introduction to Machine Learning

Course Book

Chapters 1-8

Introduction to Machine Learning

Matan Gavish and Gilad Green

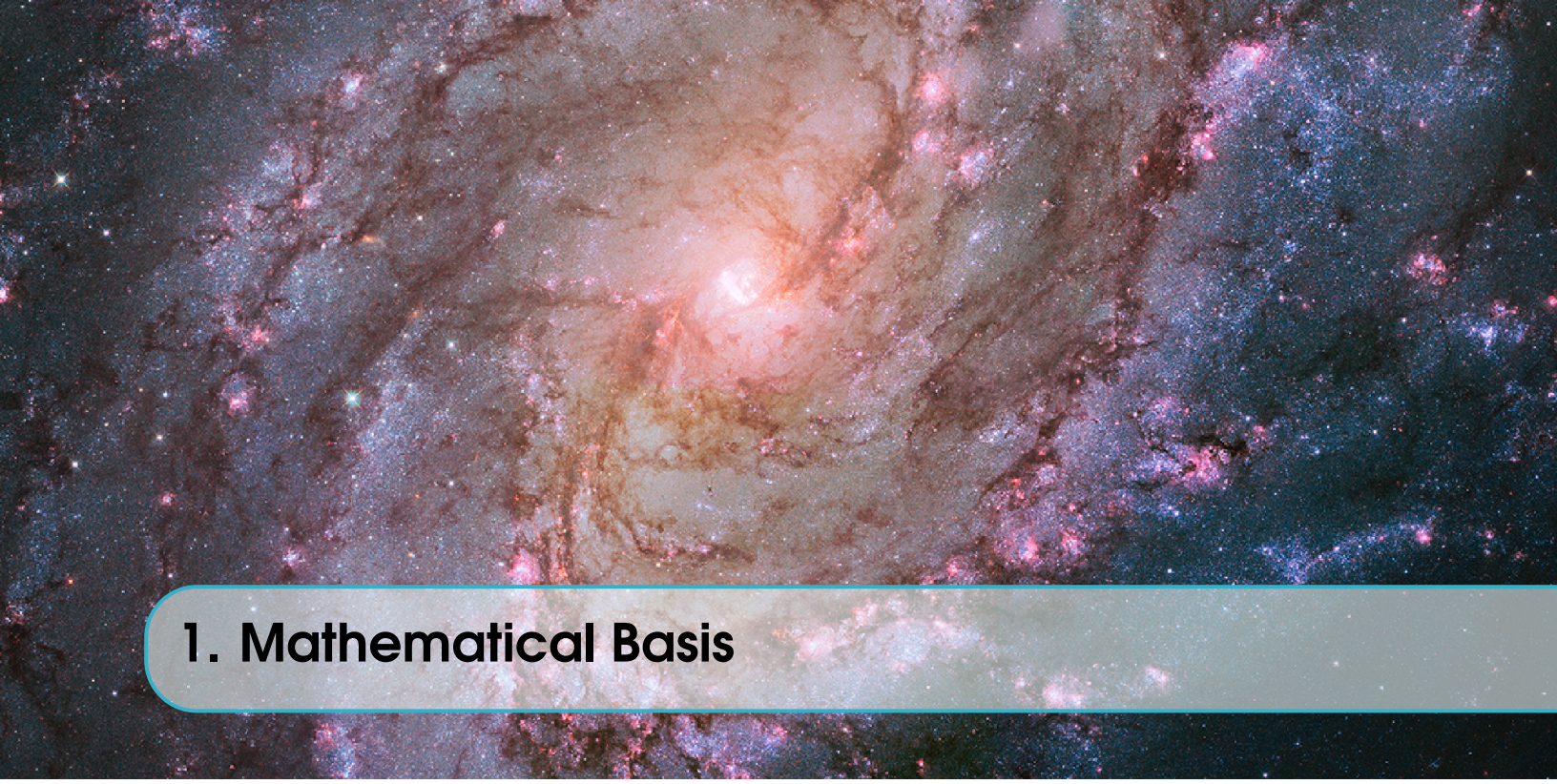
Contents

0.1 Preface

0.1.1 Notation

Field/Context	Symbol	Meaning
General Mathematics	$\mathbb{R}$	The set of real numbers
	$\mathbb{R}^d$	The set of d-dimensional vectors over the field of real numbers
	$\mathbb{R}_+$	The set of non-negative real numbers
	$\mathbb{N}$	The set of natural numbers
	$\mathbb{R}^{m\times d}$	The set of real matrices with m rows and d columns
	$\{0,\ldots,m\}$	The set of natural numbers between 0 and m
	$[m]$	The set of natural numbers between 1 and m
	$\lvert S\rvert$	The number of elements in the set S
	$a \coloneqq b$	Defining a as the experssion b
	$a \propto b$	a is of size proportional to the size of b
	$\mathbf{x},\mathbf{y},\mathbf{w}$	Column vectors
	x_i, y_i, w_i	The i-th coordinate of a vector
	$\langle \mathbf{v},\mathbf{u}\rangle$ or $\mathbf{v}^\top\mathbf{u}$	Standard inner product between $\mathbf{v}$ and $\mathbf{u}$
	$\mathbf{v}\otimes\mathbf{u}$ or $\mathbf{v}\mathbf{u}^\top$	The outer product of $\mathbf{v}$ and $\mathbf{u}$
	$\lvert\lvert\cdot\rvert\rvert_1$	$=\sum_{i=1}^d \lvert x_i\rvert$ (the ℓ_1 norm of $\mathbf{x}$)
	$\lvert\lvert\cdot\rvert\rvert_2$ or $\lvert\lvert\cdot\rvert\rvert$	$=\sqrt{\sum_{i=1}^d x_i^2}$ (the ℓ_2 norm of $\mathbf{x}$)
	$\lvert\lvert\cdot\rvert\rvert_\infty$	$= max_i \lvert x_i\rvert$ (the ℓ_∞ norm of $\mathbf{x}$)
	$A_{i,j}$	The jth element of the ith row of A
	$A^\top$	The transpose of A: $A^\top_{i,j} = A_{j,i}$
	$det\,(A)$ or $\lvert A\rvert$	The determinant of the matrix A
	$V^\perp$	The set of vectors that are perpendicular to the vector space V
	$\partial f/\partial \mathbf{x}_i$	The derivative of the function f w.r.t the ith coordinate of $\mathbf{x}$
	∇f	The gradient of the function f
	$\mathbf{J}_\mathbf{x}(f)$	The Jacobian of the function f w.r.t $\mathbf{x}$
	$\overset{!}{=}$	Expressing that we would like to equate the left side to the right side. Usually comes in the context of equating the derivative to zero and solving for the parameter in question.
	Ω	A sample space
	$\mathbb{P}$	The probabiltiy of an event or random variable
	$\mathbb{P}_\mathcal{D}$	The probability of an event or random variable w.r.t the distribution $\mathcal{D}$
	$\mathbb{E}$	The expectation of a random variable
	$x \sim \mathcal{D}$	Sampling x according to $\mathcal{D}$

Field/Context	**Symbol**	**Meaning**
Machine	m	Number of samples
Learning	d, k	Dimension of sample vector; Number of features
	$\mathcal{X}$	Domain set
	$\mathcal{Y}$	Response set
	$\mathcal{Z} = (\mathcal{X} \times \mathcal{Y})$	The product space of domain set and response set
	$S = x_1, \ldots, x_n$	A sequence of samples from domain set
	$S = z_1, \ldots, z_n$	A sequence of samples and responses from product space $\mathcal{Z}$
	$x_1, \ldots, x_m \overset{i.i.d}{\sim} \mathcal{D}$	Sampling m values independently and all according to $\mathcal{D}$
	θ	Scalar or vector representing some parameter of a distribution
	Θ	A set of parameters
	$\widehat{\theta}$	An estimator of θ
	$\mathcal{H}$	Set of functions, named hypothesis class
	$\mathcal{A}$	Learning algorithm
	$\mathcal{O}$	Big-O asymptotic notation



1. Mathematical Basis

1.1 Linear Algebra

1.1.1 Linear Transformations

Definition 1.1.1 — Linear Transformation. Let $V \in \mathbb{R}^d$ and $W \in \mathbb{R}^m$ be two vectors spaces. A function $T : V \to W$ is called a linear transformation of V into W, if $\forall u, v \in V$ and $c \in \mathbb{R}$.

- Additivity: $T(u+v) = T(u) + T(v)$
- Scalar multiplication: $T(cu) = cT(u)$

For V and W of a finite dimension, any linear transformation can be represented by a matrix A. Therefore, from now and on we will focus only on finite-dimensional spaces, and implicitly refer to the matrix representing the linear transformation.

Definition 1.1.2 — Affine Transformation. An *affine transformation* is a transformation of the form $T(u) = Au + w$, where $u \in V, w \in W$.

Notice, that by definition an affine transformation is not a linear transformation. Notice that for a linear transformation A it holds that $A \cdot 0_V = 0_W$, but in the case of an affine transformation where $0 \neq w \in W$ then $T(0_V) = A \cdot 0_V + w \neq 0_w$.

Let us define some vector spaces associated with each linear transformation

Definition 1.1.3 — Fundamental Subspaces. Let A be the matrix corresponding the linear transformation $T : V \to W$. We define the *four fundamental subspaces*:

- Kernel- (or null-) space of A as $Ker(A) := \{x \in V | Ax = 0\}$. Also denotes as $\mathcal{N}(A)$.

- Image- (or column/range-) space of A as $Im(A) \coloneqq \{w \in W | w = Ax, x \in V\}$. Also denotes as $Col(A)$ or $\mathcal{R}(A)$.
- Row space of A as $Im\left(A^\top\right) \coloneqq \left\{x \in V | x = A^\top w, w \in W\right\}$. Equivelantly it can be defined as the column space of $A^\top$ and therefore denoted as $Col\left(A^\top\right)$.
- Null space of $A^\top$ as $Ker\left(A^\top\right) \coloneqq \left\{x \in W | A^\top x = 0\right\}$. This space is also referred to as the left null space of A.

Note that by definition, $Ker(A), Row(A) \subseteq V$ and $Im(A) \subseteq W$. Using the above definitions let us gain some insights into what these vector spaces provide us with.

Definition 1.1.4 Let $A \in \mathbb{R}^{m \times d}$. The rank of A is the maximum number of linearly independent rows of A and denoted by $rank(A)$.

It holds that the rank of A equals both the dimension of the columns space and of the row space of A. As such, we refer to A being of *full rank* if and only if $rank(A) = min(m,d)$. Otherwise we say that A is rank deficient.

Definition 1.1.5 Let $A \in \mathbb{R}^{d \times d}$ be a square matrix. A is called invertible (or non-singular) if there exists a matrix $B \in \mathbb{R}^{d \times d}$ such that $AB = I_d = BA$. We denote the inverse by A^{-1}.

Claim 1.1.1 Let A be a square matrix. The following are equivalent (TFAE):
- A is invertible (non-singular)
- A is full-rank
- $Det(A) \neq 0$
- $Im(A) = \mathbb{R}^m$ (i.e., the image is the whole space)
- $ker(A) = \vec{0}$

■ **Example 1.1** Consider the following scenario: Suppose we are given a set of d linearly independent linear equations, each of the form $y_i = \sum_{j=1}^{d} \mathbf{w}_j \cdot x_{ij}$, where the $x_{i,j}$'s and y_i are given while $\mathbf{w}_j$'s are unknown. We would like to find a solution for this system of equations. That is, a coefficients vector $\mathbf{w} \in \mathbb{R}^d$ that satisfies:

$$\forall i \in [d] \; y_i = \sum_{j=1}^{d} \mathbf{w}_j \cdot x_{ij} = \mathbf{w}^\top x_i$$

Let us rearrange the equations in matrix form. Given a linear equation we will denote all it's x's by the vector $x_i \in \mathbb{R}^d$ where i denotes the numbering of the current equation. Similarly we will arrange all the y's in a vector $y \in \mathbb{R}^d$. Thus, we can represent the problem written above as follows:

$$\text{Find } \mathbf{w} \in \mathbb{R}^d \text{ such that } y = X\mathbf{w}$$

As we assumed that all linear equations are independent, the rows of X are linearly independent. Therefore, it is of full rank and there exists an invertible matrix X^{-1} such that $XX^{-1} = I$. Equipped

with this observation finding $\mathbf{w}$ is simply:

$$y = X\mathbf{w} \Rightarrow X^{-1}y = X^{-1}X\mathbf{w} \Rightarrow, \mathbf{w} = X^{-1}y$$

■

R Let us think of each vector $x_i \in \mathbb{R}^d$ as some independent observation (or sample) we have of some phenomena. Each coordinate of x_i corresponds some measurement we have of this observation. Together with this sample we are given some response value $y_i \in \mathbb{R}$. By solving for $\mathbf{w}$ we learn the relation between the x's and y's. Now suppose we are given a new sample $x \in \mathbb{R}^d$. As we already know the relation between the xs and the ys, we can predict what is the appropriate y value it achieves.
The general problem of finding such vectors is called **Regression**. In the case where the relationship is linear it is called **Linear Regression**. We will discuss linear regression in chapter 2.

1.1.2 Norms, Inner Products and Projections

In many applications in machine learning we are interested in measuring distances between vectors or sizes of vectors, and "using" a vector (or set of vectors) on another vector. For such, let us formulate these notions.

Definition 1.1.6 — Metric. A function on a set $X \subseteq \mathbb{F}^k$ $d : X \times X \to \mathbb{R}_+$ is called a metric function (or distance function) iff for any $v,u,w \in X$ it holds that:

- $d(v,u) = 0 \iff v = u$
- Symmetry: $d(v,u) = d(u,v)$
- Triangle inequality $d(v,u) \leq d(v,w) + d(w,u)$.

These conditions also imply that a metric is non-negative. As such, we also call a metric function a positive-definite function. Some common metric functions are the absolute distance or the Euclidean distance.

Exercise 1.1 Let $v,u \in \mathbb{R}^k$. Show that the absolute distance, defined as the sum of absolute element-wise subtraction between the vectors $d(v,u) := \sum |v_i - u_i|$, is a metric function. ■

Proof. Firstly, notice that for some scalars $a,b \in \mathbb{R}$ it holds that $|a-b| = 0$ iff $a = b$. Therefore d, being a sum of non-negative elements equals zero iff all elements are zero. This takes place iff $v = u$. Next, symmetry of d is achieved through symmetry of the absolute value function. Lastly, let $v,u,w \in \mathbb{R}^k$ then

$$d(v,u) = \sum |v_i - u_i| = \sum |v_i - w_i + w_i - u_i| \leq \sum |v_i - w_i| + \sum |w_i - u_i| = d(v,w) + d(w,u)$$

■

Next, let us define the notion of a *size* of a vector.

Definition 1.1.7 — Norm. A norm is a function $||\cdot|| : \mathbb{R}^d \to \mathbb{R}_+$ that satisfies the following three conditions for all $a \in \mathbb{R}$ and all $u,v \in \mathbb{R}^d$:

- Positive definite: $||v|| \geq 0$ and $||v|| = 0$ iff v is the zero vector.
- Positive homogeneity: $||av|| = |a| \cdot ||v||$.

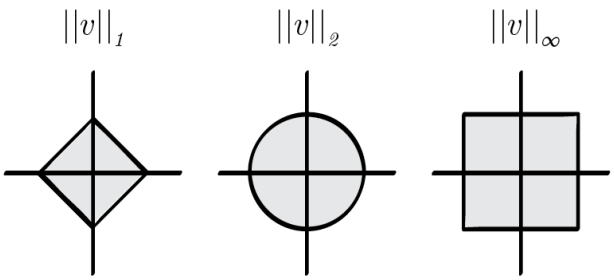
- Triangle inequality: $||v+u|| \leq ||v|| + ||u||$.

We can think of this size in the sense of vector's *distance* from the origin, under some distance function defined by the norm. A few commonly used norms are:

- Absolute norm (ℓ_1): $||v||_1 \coloneqq \sum |v_i|$.
- Euclidean norm (ℓ_2): $||v||_2 \coloneqq \sqrt{\sum x_i^2}$.
- Infinity norm: $||x||_\infty \coloneqq max_i |v_i|$.

R The absolute and Euclidean norms are part of a wider family of norms called the L_p norms defined as $||v||_p \coloneqq \left(\sum |v_i^p|\right)^{1/p}, \quad p \in \mathbb{N}$.

Definition 1.1.8 Let V be a vector space and $||\cdot||$ be a norm over this space. The unit ball of $||\cdot||$ is defined as the set of vectors such that: $B_{||\cdot||} = \{v \in V : ||v|| \leq 1\}$.



Now that we have defined the notions of distances and sizes of vectors, we want to define what it means to "apply" some vector on another.

Definition 1.1.9 — Inner Product. An inner product space is a vector space V over $\mathbb{R}$ together with a map $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$ satisfying that $\forall v, u, w \in V, \alpha \in \mathbb{R}$:

- Symmetry: $\langle v, u \rangle = \langle u, v \rangle$
- Linearity: $\langle \alpha v + w, u \rangle = \alpha \langle v, u \rangle + \langle w, u \rangle$
- Non-negativity : $\langle v, v \rangle \geq 0$ and $\langle v, v \rangle = 0 \iff v = 0$

Notice the similarity between the definition of a norm and of an inner product. In fact, given an inner-product space, we are also given a norm on this space.

Claim 1.1.2 — Induced Norm. Let H be an inner product space. Then the function $||\cdot|| : H \to \mathbb{R}_+$ is defined $\forall v \in H$ by $||v|| = \langle v, v \rangle^{\frac{1}{2}}$ is a norm on H.

Exercise 1.2 Let $v, u \in V$. Show that $\langle v, u \rangle = ||v||\,||u|| \cos\theta$, where θ is the angle between v, u. ■

Proof. Recall the Law of Cosines: in a triangle with lengths a, b, c, then

$$c^2 = a^2 + b^2 - 2ab\cos\theta$$

By applying the cosine law to the triangle defined by v and u and $v-u$ we see that:

$$||v-u||^2 = ||v||^2 + ||u||^2 - 2\,||v||\cdot||u||\cdot\cos\theta$$

On the other hand we also know that:

$$||v-u||^2 = \langle v-u, v-u\rangle = \langle v,v\rangle - 2\langle v,u\rangle + \langle u,u\rangle = ||v||^2 + ||u||^2 - 2\langle v,u\rangle$$

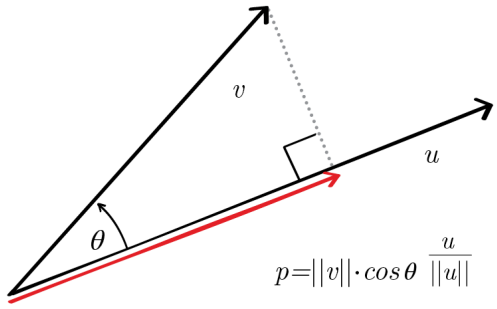
Hence, we conclude that:

$$||v||\cdot||u||\cdot\cos\theta = \langle v,u\rangle$$

■

From the above, we have an expression for the angle between two vectors, using the inner-product. We can therefore define what it means to project one vector onto the other. Using the identity of $\cos\theta$:

$$p = ||v||\cos\theta\cdot\frac{u}{||u||} = ||v||\,\frac{\langle v,u\rangle}{||v||\cdot||u||}\cdot\frac{u}{||u||} = \frac{\langle v,u\rangle}{||u||^2}\cdot u$$



Definition 1.1.10 — Vector Projection. A projection of a vector v onto a vector u, is a vector p of length $||v||\cos\theta$ in the direction of u.

Notice, that for the special case where $\theta = 90^\circ$ we get $\langle v,u\rangle = 0$. In this case we say that the vectors v,u are "orthogonal", and use the notation: $v\perp u$. If v,u are also unit vectors we say that the vectors v,u are "orthonormal" to each other.

Definition 1.1.11 Let $(V,||\cdot||)$ be a normed space. We say that $v\in V$ is a unit vector iff $||v|| = 1$.

Definition 1.1.12 An orthogonal matrix is a square matrix whose columns are unit vectors orthogonal to one another (i.e. they are orthonormal vectors) and whose rows are unit vectors orthogonal to one another.

Lemma 1.1.3 Let $A \in \mathbb{R}^{d\times d}$ orthogonal matrix, then

$$AA^\top = I = A^\top A$$

Putting together the definitions of a vector projection and orthogonal matrices we can define the notion of orthogonal projecting a vector onto some linear subspace.

Definition 1.1.13 — Outer Product. Let $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$. The outer product of u and v, denoted by $u \otimes v$ or $uv^\top$, is defined as an $m \times n$ matrix as follows:

$$(u\otimes v)_{ij} = u_i \cdot v_j, \quad u\otimes v = \begin{bmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \cdots & u_mv_n \end{bmatrix}$$

Definition 1.1.14 Let V be a k-dimensional subspace of $\mathbb{R}^d$, and let $v_1, \ldots, v_k$ be an orthonormal basis of V. Define $P = \sum_{i=1}^k v_i \otimes v_i = \sum_{i=1}^k v_i v_i^\top$. The matrix P is an ***orthogonal projection matrix*** onto the subspace V.

The following lemma summarizes some useful properties of orthogonal projection matrices.

Lemma 1.1.4 Let $v_1, \ldots, v_k$ be a set of orthonormal vectors, and let $P = \sum_{i=1}^k v_i \otimes v_i^\top = \sum_{i=1}^k v_i v_i^\top$. P has the following properties:

- P is symmetric
- $P^2 = P$
- The eigenvalues of P are either 0 or 1. $v_1, \ldots, v_k$ are the eigenvectors of P which correspond to the eigenvalue 1.
- $(I - P)P = 0$
- $\forall x \in \mathbb{R}^d$ and $\forall u \in V$, $||x - u|| \geq ||x - Px||$
- $x \in V \Rightarrow Px = x$

R The outer product of two non-zero vectors defines a matrix with rank of 1. Notice, that the orthogonal projection matrix, being a sum of such matrices, is in fact a sum of rank 1 matrices.

1.1.3 Matrix Decompositions

Matrix factorizations/decompositions are a strong tool with many theoretical as well as practical usages. It often appears in many different machine learning approaches, some of which we will encounter.

Definition 1.1.15 Let A be a square matrix. A is diagonalizable if there exists an invertible matrix P such that $P^{-1}AP$ is diagonal.

Next, we would like to see if we could represent A as the multiplication of orthogonal matrices, and a diagonal one.

Definition 1.1.16 — Eigenvector and Eigenvalue. Let A a square matrix. We say that a vector $0 \neq v \in V$ is an eigenvector of A corresponding to an eigenvalue $\lambda \in \mathbb{R}$ if $Av = \lambda v$.

Recall that any square *normal* matrix has an orthonormal basis of eigenvectors. In particular this holds for symmetric matrices:

Claim 1.1.5 Let A be a square symmetric matrix. Then there exists an orthonormal basis $u_1, ..., u_n \in \mathbb{R}^d$ of eigenvectors of A.

A diagonalizable matrix can be written as a product involving its eigenvectors and eigenvalues. This is sometimes known as the Eigenvalue Decomposition (EVD). For symmetric matrices we have:

Theorem 1.1.6 — Matrix Diagonalization - EVD. Let $A \in \mathbb{R}^{d \times d}$ be a real symmetric matrix. Then there exist an orthogonal matrix $U \in \mathbb{R}^{d \times d}$ and a diagonal matrix D such that $A = UDU^\top$ and $D_{i,i}$ are the eigenvalues of A ($i = 1..n$).

This decomposition is very useful. For example, notice that it is very easy to compute high powers of A: $A^k = UDU^\top \cdot UDU^\top \cdot UDU^\top = UD^kU^\top$. It is also easy to compute the inverse of A, if it exists: $A^{-1} = UD^{-1}U^\top$.
A serious drawback of the EVD is not every matrix can be decomposed this way. We now derive another decomposition, one which exists for any matrix - for non-symmetric and even non-square matrices.

Definition 1.1.17 Let $A \in \mathbb{R}^{m \times d}$ and let $v \in \mathbb{R}^d$, $u \in \mathbb{R}^m$ be unit vectors. We say that v, u are right- and left singular vectors of A, respectively, corresponding to a singular value $\sigma \in \mathbb{R}_+$ if $Av = \sigma u$.

Theorem 1.1.7 — Singular Values Decoposition (SVD). Let $A \in \mathbb{R}^{m \times d}$ be a real matrix. A can be written as a singular value decomposition of the form $A = U\Sigma V^\top$, where:

- The matrix $U \in \mathbb{R}^{m \times m}$ is an orthogonal matrix whose columns are the left singular vectors of A.
- The matrix $\Sigma \in \mathbb{R}^{m \times d}$ is a matrix where $\forall i \neq j \; \Sigma_{ij} = 0$ and $\Sigma_{ii} \equiv \sigma_i$ are non-negative and satisfy $\sigma_1 \geq \ldots \geq \sigma_d \geq 0$. The diagonal elements $sigma_i$ are the singular values of A.
- The matrix $V \in \mathbb{R}^{d \times d}$ is an orthogonal matrix whose columns are the right singular vectors of A.

As this decomposition exists for any real matrix it makes it very useful in many learning applications. It also has many useful properties such as:

- It provides a representation of the range (columns space) and kernel (null space) of the matrix A. The right-singular vector corresponding to vanishing singular values (i.e with value of zero) span $Ker(A)$.
- From the above we realize that the rank of A equals to the number of non-zero singular values. Looking at the singular values themselves (also referred to as spectrum) we can also derive the *effective rank* of the matrix. As performing algebraic operations are sensitive to floating-point errors, positive but very small singular values might cause a rank deficient matrix to appear of full rank. Zeroing singular values beneath a certain threshold are often assumed to be zero. Then the number of singular values over such threshold represents the effective rank of the

matrix.

- As we can derive the rank, in the case of a square matrix if all singular values are strictly larger than zero ($\sigma_d > 0$) then $dim\,(Ker\,(A)) = 0$ and the matrix is invertible.
- Based on the SVD we can define the **Moore–Penrose pseudoinverse** of a matrix $A^\dagger = V\Sigma^\dagger U^\top$ where $\Sigma^\dagger$ is obtained from Σ by replacing every non-zero singular value with its multiplicative inverse. This definition will come in handy when solving the linear regression problem (2.1.2.5).

Beyond the properties above, notice that we can also represent the SVD in a more compact manner. Suppose that $rank\,(A) = r$. This means that the number of non-zero singular values is r, and notice that $r \leq \min\{d,m\}$. When $m \leq d$ then A and Σ are both wide matrices (they have more columns than rows):

$$A = U\Sigma V^\top = \begin{bmatrix} | & & | & & | \\ u_1 & \cdots & u_r & \cdots & u_m \\ | & & | & & | \end{bmatrix} \left[\begin{array}{ccc|ccc} \sigma_1 & \cdots & 0 & & & \\ \vdots & \ddots & \vdots & & 0 & \\ 0 & \cdots & \sigma_r & & & \\ \hline & & & 0 & \cdots & 0 \\ & 0 & & \vdots & \ddots & \vdots \\ & & & 0 & \cdots & 0 \end{array}\right] \begin{bmatrix} - & v_1^\top & - \\ & \vdots & \\ - & v_r^\top & - \\ & \vdots & \\ - & v_d^\top & - \end{bmatrix}$$

Since $\sigma_{r+1}, \ldots \sigma_m$ are all zero, and any off diagonal element of Σ is zero, the left- and right singular values with indices greater than r are multiplied by zeros and do not take part in the final construction of the matrix A. Their purpose is in expanding the set of left- and right singular vectors to form a basis for $\mathbb{R}^m$ and $\mathbb{R}^d$ respectively. This means that the important information carried by the SVD about the matrix A is actually contained in a smaller $r \times r$ matrix, sometimes called the **compact SVD of** A, which we can write as:

$$A = \tilde{U}\tilde{\Sigma}\tilde{V}^\top = \overbrace{\begin{bmatrix} | & & | \\ u_1 & \cdots & u_r \\ | & & | \end{bmatrix}}^{m\times r} \overbrace{\begin{bmatrix} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_r \end{bmatrix}}^{r\times r} \overbrace{\begin{bmatrix} - & v_1^\top & - \\ & \vdots & \\ - & v_r^\top & - \end{bmatrix}}^{r\times d} \tag{1.1}$$

To avoid cluttered notations we will drop the $\tilde{\cdot}$ notation and refer to U, Σ, V in the compact form.

The two decompositions seen above and connected to one another. The following lemma the SVD of A to the EVD of $AA^\top$ and $A^\top A$. In particular, it shows that the SVD of A can be calculated in polynomial time in m and d.

Lemma 1.1.8 Let $A = U\Sigma V^\top$ be an SVD of $A \in \mathbb{R}^{m\times d}$. Then $AA^\top = U\Sigma\Sigma^\top U^\top$ is an EVD of $AA^\top$, and $A^\top A = V\Sigma^\top \Sigma V^\top$ is an EVD of $A^\top A$.

Lemma Lemma 1.1.8 therefore states that:

1. The left singular values of A (columns of U) are the eigenvectors of $AA^\top$.
2. The right singular values of A (columns of V) are the eigenvectors of $A^\top A$.
3. The singular values of A are just square root of the (shared) eigenvalues of $AA^\top$ and $A^\top A$. The eigenvalues $\sigma_1^2, \ldots, \sigma_d^2$ are the shared eigenvalues of $A^\top A$ and $AA^\top$.

R Note however, that the inverse claim is not correct. Take, for example, $A = U_1 \Sigma V^\top$ with $U_1 \equiv -U$. Both relations, $AA^\top = U\Sigma\Sigma^\top U^\top$ and $A^\top A = V\Sigma^\top \Sigma V^\top$ are still EVD's but $A \neq U\Sigma V^\top$.

1.2 Multivariate Calculus

Often when considering different machine learning techniques we will consider high dimensional functions $f : \mathbb{R}^d \to \mathbb{R}$. Usually, these functions will represent some distance measurement between our estimated prediction and the true values. As such, we would like to solve the optimization problem of minimizing this distance. Namely:

$$\underset{\mathbf{w}\in\mathbb{R}^d}{argmin}\, f(\mathbf{w})$$

1.2.1 Derivatives, Gradients and Jacobians

1.2.1.1 Derivatives

A common approach for finding such a minimizer is by computing the derivative of the function, equating to zero and solving for our parameters $\mathbf{w}$. When discussing multivariate functions we use gradients rather than scalar derivatives.

Definition 1.2.1 — Derivative. Let $f : \mathbb{R} \to \mathbb{R}$. The derivative of f at point $x \in \mathbb{R}$ is defined as

$$\frac{d}{dx} f(x) := \lim_{a\to 0} \frac{f(x+a) - f(x)}{a}$$

■ **Example 1.2 — ReLU.** Consider the **R**ectified **L**inear **U**nit function defined as the positive part of its argument: $f(x) = x^+ = \max(0, x)$. This function is in common use in the context of artificial neural networks. The derivative if this function is:

$$\frac{df(x)}{dx} = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$

■

Note that at $x = 0$ the derivative of ReLU is undefined. We will discuss ways to handle such cases in **??**. Next, let us expand the derivative definition to multivariate functions.

Definition 1.2.2 — Partial Derivative. Let $f : \mathbb{R}^d \to \mathbb{R}$. The partial derivative of f at point $\mathbf{x} \in \mathbb{R}^d$ with respect to x_i is defined as

$$\begin{aligned} \frac{\partial}{\partial x_i} f(\mathbf{x}) &:= \lim_{a\to 0} \frac{f(\mathbf{x} + a\mathbf{e}_i) - f(\mathbf{x})}{a} \\ &= \lim_{a\to 0} \frac{f(x_1, \ldots, x_i + a, \ldots, x_d) - f(x_1, \ldots, x_i, \ldots, x_d)}{a} \end{aligned}$$

where $\mathbf{e}_i$ is the i-th standard basis vector.

Namely, a partial derivative of a function is its derivative with respect to one of its variables, while all others are kept constant.

■ **Example 1.3 — Max.** For $f(\mathbf{x}) = max_i(x_1, \ldots, x_d)$ the partial derivatives of f at x_i are:

$$\frac{\partial}{\partial x_i} f(\mathbf{x}) = \begin{cases} 1 & i = argmax(x_1, \ldots, x_d) \\ 0 & i \neq argmax(x_1, \ldots, x_d) \end{cases}$$

■

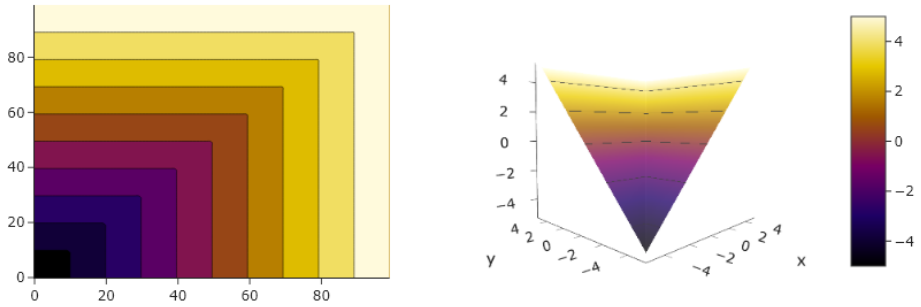


Figure 1.1: *Visualization of bi-variate max function in 3D and 2D. Chapter 1 Examples - Source Code*

■ **Example 1.4 — Polynomial.** For $f(x,y) = x^2 + xy + y^2$ the partial derivatives of f at (x_0, y_0) are

$$\frac{\partial}{\partial x} f(x_0, y_0) = 2x_0 + y_0, \quad \frac{\partial}{\partial y} f(x_0, y_0) = 2y_0 + x_0$$

■

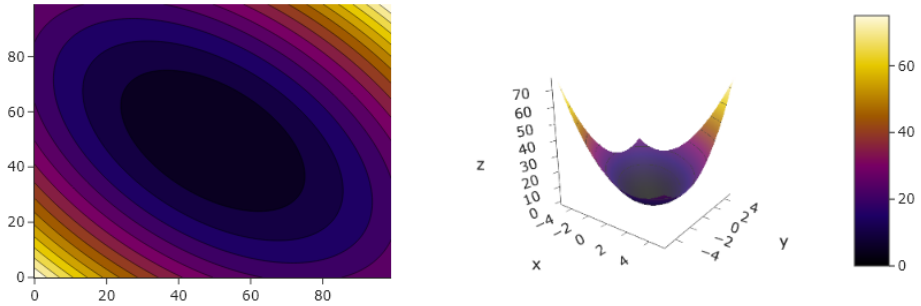


Figure 1.2: *Visualization of bi-variate polynomial of degree 2 in 3D and 2D. Chapter 1 Examples - Source Code*

1.2.1.2 Gradients

Definition 1.2.3 — Gradient. The gradient of f at $\mathbf{x}$ is the vector of partial derivatives:

$$\nabla f(\mathbf{x}) := \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_d} \right)^\top$$

■ **Example 1.5** By using the partial derivatives previously calculated of a second order bivariate polynomial $f((x,y) = x^2 + xy + y^2$, the gradient of f at $\mathbf{x}_0 = (x_0, y_0)$ is

$$\nabla f(\mathbf{x}_0) = (2x_0 + y_0, 2y_0 + x_0)^\top.$$

■

Exercise 1.3 — Linear Functional. Let $\mathbf{w} \in \mathbb{R}^d$ and $f : \mathbb{R}^d \to \mathbb{R}$ defined as $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$. Compute the gradient of f at point $\mathbf{x}$. ■

Proof. From linearity of the derivative:

$$\frac{\partial}{\partial x_j} f(\mathbf{x}) = \sum_i \frac{\partial}{\partial x_j} f(\mathbf{x})_i = \sum \frac{\partial}{\partial x_j} \mathbf{w}_i \mathbf{x}_i = \mathbf{w}_j$$

Therefore, in vector notation $\nabla f(\mathbf{x}) = \mathbf{w}$. ■

Exercise 1.4 — Norm. Let $f : \mathbb{R}^d \to \mathbb{R}$ be defined by $f(\mathbf{x}) = ||\mathbf{x}||^2$. Compute the gradient of f at point $\mathbf{x}$. ■

Proof. Similar to the previous exercise, using the linearity of the derivative then:

$$\frac{\partial}{\partial x_j} f(x) = \sum_i \frac{\partial}{\partial x_j} x_i^2 = 2x_j$$

which in vector notation can be written as: $\nabla f(\mathbf{x}) = 2\mathbf{x}$. ■

1.2.1.3 Jacobians

In different applications in machine learning, the function we are trying to minimize is a vector-valued function, namely $f : \mathbb{R}^m \to \mathbb{R}^d$.

For example, consider the following scenario. We have gathered historic information of season, time of day, latitude and longitude at different points in the world. Next, we have described some function that given these parameters states the temperature and air-pressure of that location and time. We want to find the the extrema points of this function. In this scenario, the described function gets fours parameters and returns two outputs $f : \mathbb{R}^4 \to \mathbb{R}^2$. In order to find the extrema points we need to generalize the gradient definition to vector-valued functions.

Definition 1.2.4 — Jacobian. Let $\mathbf{f} : \mathbb{R}^d \to \mathbb{R}^m$ where $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))^\top$. The Jacobian of f is the $m \times d$ matrix of all partial derivatives:

$$J_{\mathbf{x}}(\mathbf{f}) := \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_d} \\ \vdots & & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_d} \end{bmatrix}$$

■ **Example 1.6** Let us revisit 1.5 where $f:\mathbb{R}^2\to\mathbb{R}$ defined as $f(\mathbf{x})=x_1^2+x_2^2$. The Jacobian of f is:

$$J_x(\mathbf{f})=\begin{bmatrix}\frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2}\end{bmatrix}=[2x_1,2x_2]=\nabla f(\mathbf{x})^\top$$

■

Notice that for any function where $k=1$ the Jacobian is in fact the transposed gradient vector: $J_{\mathbf{x}}(f)=\nabla f(\mathbf{x})^\top$.

Exercise 1.5 Let $f:\mathbb{R}^d\to\mathbb{R}^m$ defined as $f(\mathbf{x})=A\mathbf{x}$ where $A\in\mathbb{R}^{m\times d}$. Find the Jacobian of f: $J_{\mathbf{x}}(f)$). ■

Proof. Define the set of functions computing each coordinate in the output vector $\forall i\in[m]\quad f_i(\mathbf{x})=A_i^\top\mathbf{x}$. Then the jacobian of f is comprised of the gradients of $f_1,\ldots,f_m$ as rows. Notice that we have already computed the gradient of linear functionals in 1.3 so:

$$J_{\mathbf{x}}(f)=\begin{bmatrix}\nabla f_1(\mathbf{x})^\top\\ \vdots\\ \nabla f_d(\mathbf{x})^\top\end{bmatrix}=\begin{bmatrix}-A_1-\\ \vdots\\ -A_m-\end{bmatrix}=A$$

■

1.2.1.4 Chain Rules

Theorem 1.2.1 — Chain Rule - Univariate. Let $f:\mathbb{R}\to\mathbb{R}$ and $g:\mathbb{R}\to\mathbb{R}$ be two differential functions, then the derivative of the composite $f\circ g$ is:

$$(f\circ g)'\coloneqq(f'\circ g)\cdot g'$$

Namely, for $h(x)=f(g(x))$ then $\forall x\in\mathbb{R}\ h'(x)=f'(g(x))\cdot g'(x)$.

Theorem 1.2.2 — Chain Rule - Multivariate. Let $f:\mathbb{R}^d\to\mathbb{R}^m$ and $g:\mathbb{R}^k\to\mathbb{R}^d$. The Jacobian of the composition $(f\circ g):\mathbb{R}^k\to\mathbb{R}^m$ at $\mathbf{x}$ is

$$J_{\mathbf{x}}(f\circ g)=J_{g(\mathbf{x})}(f)J_{\mathbf{x}}(g)\coloneqq\begin{bmatrix}\frac{\partial f_1(g(\mathbf{x}))}{\partial g_1(\mathbf{x})} & \cdots & \frac{\partial f_1(g(\mathbf{x}))}{\partial g_d(\mathbf{x})}\\ \vdots & & \vdots\\ \frac{\partial f_m(g(\mathbf{x}))}{\partial g_1(\mathbf{x})} & \cdots & \frac{\partial f_m(g(\mathbf{x}))}{\partial g_d(\mathbf{x})}\end{bmatrix}=\begin{bmatrix}\frac{\partial g_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_1(\mathbf{x})}{\partial x_k}\\ \vdots & & \vdots\\ \frac{\partial g_d(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial g_d(\mathbf{x})}{\partial x_k}\end{bmatrix}$$

In element-wise notation:

$$J_{\mathbf{x}}(f\circ g)_{i,j}\coloneqq\sum_l\frac{\partial f_i(g(\mathbf{x}))}{\partial g_l(\mathbf{x})}\frac{\partial g_l(\mathbf{x})}{\partial x_j}$$

Exercise 1.6 Let $f(\mathbf{x})=||\mathbf{x}||^2$ and $g(\mathbf{x})=A\mathbf{x}$ for some matrix $A\in\mathbb{R}^{m\times d}$. Calculate the jacobian of $f\circ g$. ■

Proof. From the previous theorem we know that:

$$J_{\mathbf{x}}(f\circ g)=J_{g(\mathbf{x})}(f)J_{\mathbf{x}}(g)$$

As g a linear transformation we have see in 1.5 that $J_{\mathbf{x}}(g) = A$. Notice, that as $Im(f) \subseteq \mathbb{R}$, the jacobian of f equals to the transpose of its gradient. As seen in 1.4, $J_{g(\mathbf{x})}(f) = (2g(\mathbf{x}))^{\top}$. Therefore:

$$J_{\mathbf{x}}(f \circ g) = 2\mathbf{x}^{\top}A^{\top}A$$

■

Next, let us calculate the gradient of the following function: $f(\mathbf{x}) = \frac{1}{2}||A\mathbf{x}-\mathbf{y}||^2$. Applying the chain rule then:

$$\begin{aligned} \nabla f(\mathbf{x}) &= \frac{\partial}{\partial \mathbf{x}}\frac{1}{2}||A\mathbf{x}-\mathbf{y}||^2 \\ &= \frac{1}{2}\frac{\partial}{\partial \mathbf{x}}\left(\mathbf{x}^{\top}A^{\top}A\mathbf{x} - 2\mathbf{y}^{\top}A\mathbf{x} + \mathbf{y}^{\top}y\right) \\ &= A^{\top}A\mathbf{x} - A^{\top}y \\ &= A^{\top}(A\mathbf{x}-\mathbf{y}) \end{aligned} \tag{1.2}$$

This function is known as the Mean Square Error (MSE) and in Figure 2.2 we will use the above derivation in order to find the values of $\mathbf{x}$ that minimize f as above.

■ **Example 1.7 — Soft-Max.** The softmax function defined over $S : \mathbb{R}^d \to [0,1]^d$ returns a vector that its coordinates sum up to 1. It is defined by

$$S(\mathbf{a})_j = \frac{e^{a_j}}{\sum_{k=1}^{N} e^{a_k}}$$

As the coefficients a_j are in the power of the exponent function, all outputted values are strictly positive. Moreover, since the numerator appears in the denominator summed up with some other positive numbers, $S_j < 1$. Therefore, it is in the range $(0,1)$. For example, the 3-element vector $(1,2,5)$ gets transformed into $(0.02, 0.05, 0.93)$. The order of elements by relative size is preserved, and they add up to 1.0. In addition, the third element is now farther away from the first two, Its softmax value is dominating the overall slice of size 1.0 in the output.

Intuitively, the *softmax* function is a "soft" version of the *argmax* function. Instead of just selecting one maximal element, softmax breaks the vector up into segments with the maximal input element getting a proportionally larger chunk, but the other elements getting some of it as well. Softmax is often used in neural networks, to map the non-normalized output to a probability distribution over predicted output classes.

Let us calculate the derivative of the softmax function. Denote $g_{i(\mathbf{a})} := e^{a_i}$ and $h(\mathbf{a}) := \sum_{k=1}^{N} g_k(\mathbf{a})$. So:

$$\frac{\partial S_i}{\partial a_j} = \frac{\partial}{\partial a_j}\frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} = \frac{\partial}{\partial a_j}\frac{g_i}{h}$$

Note that for any a_j the derivative of h is e^{a_j}. In the case of g_i, when deriving with respect to a_j we get that the derivative is e^{a_j} only if $i = j$. Otherwise, the derivative is 0. Therefore, the derivative of S_i in the case where $i = j$ is:

$$\frac{\partial}{\partial a_j}\frac{e^{a_i}}{\sum_{k=1}^{N} e^{a_k}} = \frac{e^{a_i}(\sum_{k=1}^{N} e^{a_k}) - e^{a_i}e^{a_j}}{(\sum_{k=1}^{N} e^{a_k})^2} = \frac{e^{a_i}}{(\sum_{k=1}^{N} e^{a_k})} \cdot \frac{(\sum_{k=1}^{N} e^{a_k}) - e^{a_j}}{(\sum_{k=1}^{N} e^{a_k})} = S_i(1 - S_j)$$

It is left to show the derivative in the case where $i \neq j$. ■

1.2.2 Function Approximations

1.2.2.1 First Order Approximation

As motivated in the begining of this section, we are often interested in finding the minimizers of some multivariate objective function. Many times, these functions are very hard, or even impossible, to solve analytically. In such cases we might consider approximating the true function with a simpler one that we are able to solve.

Consider a function $f : \mathbb{R} \to \mathbb{R}$ and recall the definition of the Taylor series:

$$T(x_0 + x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!} x^n = f(x_0) + f'(x_0)\, x + \frac{1}{2} f''(x_0)\, x^2 + \dots$$

A linear approximation (or first order approximation) is an approximation of a general function using a linear function. For a twice continuously differentiable function $f : \mathbb{R} \to \mathbb{R}$, Taylor's theorem implies that for small x

$$f(x_0 + x) \approx f(x_0) + f'(x_0)\, x$$

We can now extend this theorem to define linear approximations of multivariate functions.

Definition 1.2.5 — Linear Approximation. Let $f : \mathbb{R}^d \to \mathbb{R}$ and $\mathbf{p}_0 \in \mathbb{R}^d$. The linear approximation of f for every $\mathbf{p}$ near $\mathbf{p}_0$ is defined as

$$f(\mathbf{p}_0) + \langle \nabla f(\mathbf{p}_0), \mathbf{p} - \mathbf{p}_0 \rangle$$

Equivalently, if we treat $\mathbf{p}$ as the deviation from $\mathbf{p}_0$ then:

$$f(\mathbf{p}_0 + \mathbf{p}) \approx f(\mathbf{p}_0) + \langle \nabla f(\mathbf{p}_0), \mathbf{p} \rangle$$

So if for example, we consider a bivariate function $f : \mathbb{R}^2 \to \mathbb{R}$, the linear approximation of f near the point (x_0, y_0) is:

$$f(x_0 + x, y_0 + y) \approx f(x_0, y_0) + x \cdot \frac{\partial f(x_0, y_0)}{\partial x} + y \cdot \frac{\partial f(x_0, y_0)}{\partial y}$$

Now, if f is a linear function, intuition dictates that the linear approximation of f would be the function itself. Let $\mathbf{b} \in \mathbb{R}^d$ and let $f : \mathbb{R}^d \to \mathbb{R}$ be defined by $f(\mathbf{x}) = \mathbf{b}^\top \mathbf{x}$. Then, the linear approximation of f at $\mathbf{p}_0$ is

$$\begin{aligned} f(\mathbf{p}_0) + \langle \nabla f(\mathbf{p}_0), \mathbf{p} - \mathbf{p}_0 \rangle &= \mathbf{b}^\top \mathbf{p}_0 + \langle \mathbf{b}, \mathbf{p} - \mathbf{p}_0 \rangle \\ &= \mathbf{b}^\top (\mathbf{p}_0 + \mathbf{p} - \mathbf{p}_0) = f(\mathbf{p}) \end{aligned}$$

Exercise 1.7 Let $f : R^2 \to \mathbb{R}$ be defined by $f(x, y) = \sqrt{x^2 + y^2}$. Calculate the linear approximation of f near $(3, 4)$. ∎

Proof. We begin with expressing the gradient of f. So, the partial derivative of f with respect to first argument at point x is:

$$\frac{\partial}{\partial x} = f(x_0, y_0) = 2x_0 \cdots \frac{1}{2\sqrt{x_0^2 + y_0^2}}$$

Therefore the gradient of f is $\nabla f(\mathbf{x}) = \left(\frac{x_0}{\sqrt{x_0^2+y_0^2}}, \frac{y_0}{\sqrt{x_0^2+y_0^2}}\right)^\top$. So for a point (x,y) in the vicinity of $(3,4)$ the linear approximation is:

$$f(3+x,4+y) \approx 5+\frac{3}{5}x+\frac{4}{5}y$$

If for example $x=0.1, y=0.2$ then $f(3+0.1,4+0.2) = 5.2201.. \approx 5.22 = 5+3/5\cdot 0.1+4/5\cdot 0.2$ ■

Another use of first order approximations is as follows. Suppose we investigate some objective function $f:\mathbb{R}^d \to \mathbb{R}$ at point $\mathbf{p}_0 \in \mathbb{R}^d$. Now, we would like to "take a step" in $\mathbb{R}^d$ in the direction where f grows with the fastest rate. That is, we are searching for the direction in space, that if we move in, f grows the most. How can we find such direction?

Recall that $f(\mathbf{p}_0+\mathbf{p}) \approx f(\mathbf{p}_0)+\nabla f(\mathbf{p}_0)\cdot\mathbf{p}$. In addition, the angle between $\nabla f(\mathbf{p}_0)$ and $\mathbf{p}$ is $\nabla f(\mathbf{p}_0)\cdot\mathbf{p} = ||\nabla f(\mathbf{p}_0)||\cdot||\mathbf{p}||\cos\theta$. As we are only interested in the direction in which to step, and not the step size, let us assume that $||(\mathbf{p})|| = 1$. Since $\cos x \in [-1,1]$, the direction that maximizes f is $\mathbf{p}_{max} := \nabla f(\mathbf{p}_0)/||\nabla(\mathbf{p}_0)||$. Similarly the direction that minimizes f is $\mathbf{p}_{min} := -\nabla f(\mathbf{p}_0)/||\nabla(\mathbf{p}_0)||$. So, if we adapt a step-wise approach, for sufficiently small enough steps sizes, for maximizing/minimizing f, moving in the direction of the gradient or opposite to the gradient is a good approach. We will revisit this concept in **??**.

1.2.2.2 Second Order Approximation

The gradient (or Jacobian) of a function is important for first order approximations of a function. To get a second order approximation we consider the second derivative of the function.

Definition 1.2.6 — Hessian. Let $f:\mathbb{R}^d \to \mathbb{R}$ be a twice differential function. The *Hessian* matrix H of f is the square matrix of second derivative:

$$H := \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \cdots & \frac{\partial^2 f}{\partial^2 x_d} \end{bmatrix}$$

■ **Example 1.8** Let us calculate the Hessian of the polynomial function $f(x_1,x_2) = x_1^2+x_1x_2+x_2^2$. We begin with calculating the first partial derivatives of f:

$$\frac{\partial f(x_1,x_2)}{\partial x_i} = 2x_i+x_j \quad i\in\{1,2\}, j\neq i$$

Next, we calculate the derivative a second time with respect to each of the parameters:

$$\frac{\partial^2 f(x_1,x_2)}{\partial x_i \partial x_j} = \begin{cases} 2 & x_i = x_j \\ 1 & x_i \neq x_i \end{cases}$$

We therefore conclude that the Hessian of f is :

$$H = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

■

By using the Hessian matrix in the second order term of the Taylor series then:

$$f(\mathbf{p}+\mathbf{p_0}) \approx f(\mathbf{p_0}) + \nabla f(\mathbf{p_0})^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top H_{f(\mathbf{p_0})}\mathbf{p} \tag{1.3}$$

■ **Example 1.9** Returning to the second order polynomial function defined in 1.8 then for any point $\mathbf{p} = (x,y)^\top$ near point $\mathbf{p}+\mathbf{0} = (x_0,y_0)$ the first order approximation is:

$$\begin{aligned} f(\mathbf{p_0}+\mathbf{p}) &\approx f(\mathbf{p_0}) + \nabla f(\mathbf{p_0})^\top \mathbf{p} \\ &= x_0^2 + x_0y_0 + y_0^2 + \begin{bmatrix} 2x_0+y_0 \\ 2y_0+x_0 \end{bmatrix}^\top \begin{bmatrix} x \\ y \end{bmatrix} \\ &= x_0^2 + x_0y_0 + y_0^2 + 2x_0x + y_0x + 2y_0y + x_0y \end{aligned}$$

Now, let us add the second order approximation:

$$\begin{aligned} f(\mathbf{p_0}+\mathbf{p}) &\approx x_0^2 + x_0y_0 + y_0^2 + 2x_0x + y_0x + 2y_0y + x_0y + \tfrac{1}{2}\begin{bmatrix} x \\ y \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\ &= x_0^2 + x_0y_0 + y_0^2 + 2x_0x + y_0x + 2y_0y + x_0y + +x^2 + yx + y^2 \\ &= (x_0+x)^2 + (x_0+x)(y_0+y) + (y_0+y)^2 \end{aligned}$$

Notice that since f is a second order polynomial then the calculated value is the exact value of the function at $\mathbf{p_0}+\mathbf{p}$. ■

1.2.3 Convexity

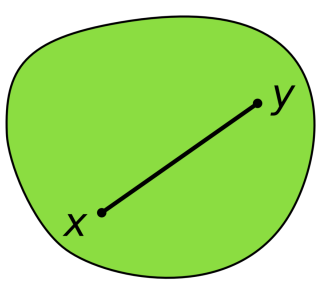
1.2.3.1 Convex Sets and Functions

Definition 1.2.7 Let V be a vector space. A set $C \subseteq V$ is called *convex* if and only if for any two vectors $u, v \in C$, and any scalar $\alpha \in [0,1]$, $\alpha u + (1-\alpha)v \in C$.

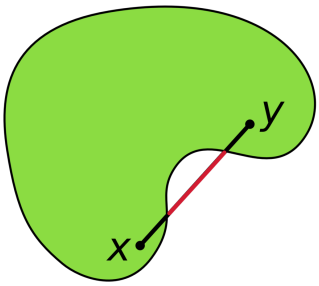
Given two vectors u, v, we can re-write the above definition as the line segment between the two vectors $u, u+v$:

$$L := \left\{ u + \frac{1-\alpha}{\alpha} v \mid \alpha \in [0,1] \right\} \tag{1.4}$$

This means that geometrically a set C is convex if and only if the line segment joining u and $u+v$ is contained in C.



(a) Illustration of a convex set with line segment contained in set.



(b) Illustration of a non-convex set as line segment not contained in set.

■ **Example 1.10 — Unit Balls.** The unit ball $B \coloneqq \{v \in V : ||v|| \leq 1\}$ is a convex set. For any two $u, v \in B$, and $\alpha \in [0,1]$, the triangle inequality implies that

$$\begin{aligned} \|\alpha u + (1-\alpha v)\| &\leq \|\alpha u\| + \|(1-\alpha)v\| \\ &= \alpha\|u\| + (1-\alpha)\|v\| \\ &\leq \alpha + 1 - \alpha \\ &= 1 \end{aligned}$$

■

■ **Example 1.11 — PSD Matrices.** The set of positive semi-definite matrices (PSD) is a convex set. Given any two PSD matrices $M, N \in \mathbb{R}^{d\times d}$ and $\alpha \in [0,1]$ then:

$$\forall \mathbf{x} \in \mathbb{R}^d \quad \mathbf{x}^\top (\alpha M + (1-\alpha)N)\mathbf{x} = \alpha \cdot \mathbf{x}^\top M\mathbf{x} + (1-\alpha)\cdot \mathbf{x}^\top Nx$$

As M, N are PSD and $\alpha \in [0,1]$ then the above is non-negative and therefore the convex combination of M, N is also PSD. Since and convex combination of M, N is PSD we conclude that the set of PSD matrices is a convex set. ■

Given the definition of a convex set we are able to derive closure properties of convex sets:

Claim 1.2.3 Closure properties of convex sets:

1. Let C_1, C_2 be two convex sets. The vector sum $C_1 + C_2 \coloneqq \{c_1 + c_2 | c_1 \in C_1, c_2 \in C_2\}$ is a convex set.
2. Let C be a convex set and $\lambda \in \mathbb{R}$. The set $\lambda C \coloneqq \{\lambda c | c \in C\}$ is a convex set.
3. Let $C_1, \ldots, C_m$ be a collection of convex sets then their intersection $\bigcap_{i\in[m]} C_i$ is a convex set.

Definition 1.2.8 Let V be a vector space and let C be a convex set. A function $f : C \to \mathbb{R}$ is called convex if for every $u, v \in C$ and every $\alpha \in [0,1]$,

$$f(\alpha u + (1-\alpha)v) \leq \alpha f(u) + (1-\alpha) f(v)$$

The function f is *strictly convex* if the above inequality holds strictly for every $u \neq v \in C$ and $\alpha \in [0,1]$.

That is, a function is called convex if and only if the line segment between any two points on the graph is above the graph. Another way to understand a convex function is by the notion of the epigraph.

Definition 1.2.9 Let $f : C \to \mathbb{R}$. The *epigraph* of f is the set $epi(f) \coloneqq \{(u,t) \mid f(u) \leq t\}$

Claim 1.2.4 A function $f : C \to \mathbb{R}$ defined over a convex set C is convex if and only if its *epigraph* is a convex set.

■ **Example 1.12 — Affine Transformations.** Let V be a linear subspace and $f : V \to \mathbb{R}$ defined by

$f(u) = u^\top w + b$ for $w \in \mathbb{R}^d, b \in \mathbb{R}$. Then f is a convex function: Let $u, v \in V$ and let $\alpha \in [0,1]$.

$$\begin{aligned} f(\alpha u + (1-\alpha)v) &= \langle w, \alpha u + (1-\alpha)v \rangle + b \\ &= \alpha\left(u^\top w + b\right) + (1-\alpha)\left(v^\top w + b\right) \\ &= \alpha f(u) + (1-\alpha) f(v) \end{aligned}$$

■

Similar to the closure properties of convex sets, there are several closure properties of convex functions. Some of these are presented in the following claim.

Claim 1.2.5 Closure properties of convex functions:

1. Let $f : \mathbb{R}^m \to \mathbb{R}$ be a convex function and $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$. The function $g : \mathbb{R}^n \to \mathbb{R}$ given by $g(u) = f(Au + b)$ is a convex function.
2. Let $f_i : V \to \mathbb{R}$ for $i = 1, \ldots, m$ be a set of convex functions and $\alpha_1, \ldots, \alpha_m$ positive scalars. The function $g(u) = \sum_{i=1}^m \alpha_i f_i(u)$ is a convex function.
3. Let $f_i : V \to \mathbb{R}$ for $i \in 1, \ldots, m$ be a set of convex functions then the function $g(u) = \sup_{i \in [m]} f_i(u)$ is a convex function.

■ **Example 1.13** Consider the function $f : \mathbb{R}^d \to \mathbb{R}$ that is defined by $\mathrm{f}(\mathbf{w}) = \left(\mathbf{x}^\top \mathbf{w} - y\right)^2$ for some $(\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R}$. Notice that by the Lemma 1.2.5 this function is convex:

- The function $\mathbf{w} \mapsto \mathbf{x}^\top \mathbf{w} - y$ is an affine function and is therefore convex.
- The scalar function $a \mapsto a^2$ is a convex function.
- From the first closure property we defive that f is a convex function.

■

1.2.3.2 High Order Conditions For Convexity

Theorem 1.2.6 — First order condition. A differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if the following inequality holds for any two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

$$f(y) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x})$$

■ **Example 1.14** Let $\mathbf{c} \in \mathbb{R}^d$ and define $f : \mathbb{R}^d \to \mathbb{R}$ by: $f(\mathbf{x}) = \mathbf{c}^\top \mathbf{x}$. f is convex, since for any $\mathbf{x} \in \mathbb{R}^d$ we have: $\nabla f(x) = \mathbf{c}$, so:

$$\forall \mathbf{y} \in \mathbb{R}^d \quad f(\mathbf{y}) = \mathbf{c}^\top \mathbf{y} = \mathbf{c}^\top \mathbf{x} + \mathbf{c}^\top (\mathbf{y} - \mathbf{x}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x})$$

■

Theorem 1.2.7 — Second order condition. A twice differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only if for any point $\mathbf{x} \in \mathbb{R}^d$, the Hessian of f evaluated at $\mathbf{x}$ is PSD: $\nabla^2 f(\mathbf{x}) \succeq 0$

■ **Example 1.15** Let $A \in \mathbb{R}^{d \times d}$ be a symmetric matrix and define $f(\mathbf{x}) = \mathbf{x}^\top A \mathbf{x}$. Let us compute the first and second derivations of f.

Denote $g(\mathbf{x}) = A\mathbf{x}$ and $h(\mathbf{x}) = \mathbf{x}$ so we can write f as $f \equiv h^\top g$. By the product rule for dot products of two vectors:

$$\nabla f(\mathbf{x}) = \nabla h(\mathbf{x})^\top g(\mathbf{x}) + h(\mathbf{x})^\top \nabla g(\mathbf{x})$$

Since the gradient of g by $\mathbf{x}$ equals to A then:

$$\nabla f(\mathbf{x}) = A\mathbf{x} + \mathbf{x}^\top A = \left(A + A^\top\right)\mathbf{x}$$

As A is symmetric then $\nabla f(\mathbf{x}) = 2A\mathbf{x}$. Next let us compute the second derivative:

$$\nabla^2 f(\mathbf{x}) = \frac{\partial^2 f}{\partial^2 \mathbf{x}} = \frac{\partial 2A\mathbf{x}}{\partial \mathbf{x}} = 2A$$

Using Lemma 1.2.7 we conclude that f is convex iff $A \succeq 0$. That is iff A is a PSD matrix. ∎

1.3 Probability and Statistics

A great deal of the machine learning principles are based on concepts from probability theory and statistics. As we will encounter later on, we are often facing a scenario where for the task in hand we:

- Define some parameter (or set of parameters) of interest.
- Define some objective function that depends on this parameter.
- Have some data relevant to the task, from which we try to **estimate** the **parameter** by optimizing the **objective function**.

Consider the following task. We are preparing the traditional "Introduction to Machine Learning Hackathon" and expecting 500 hungry students. For this event we want to order enough pizzas for all the participants. How should we know how many pizzas to order? Let us denote the average number of pizza slices each participant would like to eat by μ. In statistics, this average is called the *population mean*, meaning simply that it refers to an average over the whole group which is under consideration. In this case, all N participants.

Then, a reasonable guess for the number of pizza slices we should order is $X = N \times \mu$. As we do not know μ, and it would take too long to ask each of the participants how many pizza slices they would like to have, we must come up with some way to **estimate** a value of μ. To do so, let us *sample independently* participants: we randomly choose one, call and ask how many pizza slices they would like to eat. We write the answer down as x_1. We then randomly call another one (which could accidentally be the same one) and write the second answer as x_2. We continue m times. Now, to estimate the population mean, μ, we use the average of the answers acquired. In statistics this is referred to as the *sample mean* and is given by

$$\widehat{\mu} \equiv \frac{1}{m}\sum_{i=1}^{m} x_i.$$

Namely, the empirical average of the parameter in question. Finally, we *predict* that we will need $T = N \times \widehat{\mu}$ pizza slices. The $\widehat{\mu}$ (hat) symbol indicates that this is an estimator of the parameter μ.

Another question is how many participants should we contact? Intuitively, if we ask a very small set of participants our estimation of $\widehat{\mu}$ (and therefore our prediction of T would be very wrong. As we

increase the number of participants asked, the *sample size*, we will get a more accurate estimation of $\widehat{\mu}$. To answer how accurate the estimation, let us suppose that we know the true value of μ. Then we should look at the difference between our estimation and the true value, and how it changes as m changes. We can then choose an accuracy parameter $\varepsilon > 0$ and require that $|\widehat{\mu} - \mu| \leq \varepsilon$. By solving for m we will get a bound on the number of participants we should ask.

1.3.1 Fundamental Definitions

1.3.1.1 Probability Space

The basic object in probability theory is that of a *probability space* which consists of two ingredients: the *sample space* and the *probability function*.

Definition 1.3.1 A *sample space* Ω is a set that contains all possible outcomes. $\omega \in \Omega$ denotes a single outcome.

For example the probability space of m coin tosses can be defined as $\Omega = \{H,T\}^m$ or the probability space of coin tosses until heads comes up can be defined as $\Omega = \left\{T^{n-1}H : n \in \mathbb{N}\right\}$. We can also define more complex probability spaces such as the height, weight, blood pressure and sex of patients with a medical condition, as well as the success or failure of an experimental drug: $\Omega = [0,\infty) \times [0,\infty) \times [0,\infty) \times \{M,F\} \times \{Yes,No\}$. As we will see, we often model machine learning tasks with some probability space (even if we do not explicitly define it).

Definition 1.3.2 An *event* A is any subset of possible outcomes, $A \subseteq \Omega$.

Definition 1.3.3 A *probability space* is a tuple [a] $(\Omega, \mathcal{D})$ where Ω is a *sample space* and $\mathcal{D} : 2^{\Omega} \to \mathbb{R}$ is a probability function such that

1. $\mathcal{D}(\Omega) = 1$
2. for all $\omega \in \Omega, \mathcal{D}(\omega) \in [0,1]$
3. for all $A,B \subseteq \Omega$ such that $A \cap B = \emptyset$, we have $\mathcal{D}(A \cup B) = \mathcal{D}(A) + \mathcal{D}(B)$.

[a] For our needs this simple and intuitive definition is sufficient, though, richer definitions exist.

■ **Example 1.16** Suppose we throw two fair dice, then $\Omega = \{1,\ldots,6\}^2$ and $\mathcal{D}((i,j)) = \frac{1}{36}$ ■

Claim 1.3.1 For all $A,B \subseteq \Omega$: $\mathcal{D}(A \cup B) = \mathcal{D}(A) + \mathcal{D}(B) - \mathcal{D}(A \cap B)$

Proof. Let us notice that for both events A,B it holds that:

$$A = (A \setminus B) \cup (A \cap B)$$
$$B = (B \setminus A) \cup (A \cap B)$$

and that $A \cup B = (A \setminus B) \cup (B \setminus A) \cup (A \cap B)$. Therefore

$$\begin{aligned} \mathcal{D}(A \cup B) &= \mathcal{D}(A \setminus B) + \mathcal{D}(B \setminus A) + \mathcal{D}(A \cap B) \\ &= \mathcal{D}(A) - \mathcal{D}(A \cap B) + \mathcal{D}(B) - \cancel{\mathcal{D}(A \cap B)} + \cancel{\mathcal{D}(A \cap B)} \end{aligned}$$

■

Definition 1.3.4 Events $A, B \subseteq \Omega$ are said to be *independent* if the occurrence of one does not affect the probability of occurrence of the other, namely:

$$\mathcal{D}(A \cap B) = \mathcal{D}(A) \cdot \mathcal{D}(B)$$

Exercise 1.8 Show that if A and B are independent then A and B^c $(\Omega \setminus B)$ are also independent. ■

Proof. As $\mathcal{D}(A) = \mathcal{D}(A \cap B) + \mathcal{D}(A \cap B^c)$ Then $\mathcal{D}(A \cap B^c) = \mathcal{D}(A)(1 - \mathcal{D}(B)) = \mathcal{D}(A) \cdot \mathcal{D}(B^c)$ ■

Theorem 1.3.2 — The union bound. Let $(\Omega, \mathcal{D})$ be a probability space. The probability function is *sub-additive*, i.e., for any sequence (A_k) of events,

$$\mathcal{D}\left(\cup_{k=1}^{\infty} A_k\right) \leq \sum_{k=1}^{\infty} \mathcal{D}\left(A_k\right)$$

Proof. Let $B_1 = A_1$. For each $k \in \{2, 3, \ldots\}$, let $B_k = A_k \setminus \cup_{i=1}^{k-1} A_i$. Note that the $B_1, \ldots, B_k$ are disjoint, and that $\cup_{i=1}^{k} A_i = \cup_{i=1}^{k} B_i$. Also, since $B_k \subseteq A_k$, $\mathcal{D}(B_k) \leq \mathcal{D}(A_k)$ for every $k \in \mathbb{N}$. It follows that:

$$\mathcal{D}\left(\cup_{k=1}^{\infty} A_k\right) = \mathcal{D}\left(\cup_{k=1}^{\infty} B_k\right) = \sum_{k=1}^{\infty} \mathcal{D}\left(B_k\right) \leq \sum_{k=1}^{\infty} \mathcal{D}\left(A_k\right)$$

■

1.3.1.2 Random Variables

So far we have asked questions like: does an event happen or not and what is the probability of the event. How would we be able to model different questions such as, "how much". For example, suppose if a coin flip turns head we get a dollar, we can ask how many dollars would we get after 3 rounds?

In order to answer this question, we should identify the sample space, which is

$$\Omega = \{HHH, HHT, HTH, THH, HTT, THT, TTH, TTT\}$$

Then, we would need to define a reward function that will quantify ones's profit for each outcome of the experiment ($\omega \in \Omega$): $X(HHH) = 3$, $X(HHT) = 2$ and so on. This quantifying function X is called a *random variable* (RV).

Definition 1.3.5 Given a probability space $(\Omega, \mathcal{D})$, a real-valued *random variable (RV)* is a function $X : \Omega \to \mathbb{R}$.

Definition 1.3.6 Let Ω be a discrete probability space and X a random variable over Ω. The ***probability mass function (PMF)*** of X is defined as:

$$\mathcal{D}(\{X = x\}) = \sum_{\omega : X(\omega) = x} \mathcal{D}(\omega)$$

Instead of writing $\mathcal{D}(\{X=x\})$ we can simplify notation as $\mathcal{D}_X(\{x\})$, and further simply as $\mathcal{D}(x)$. This simplified notation, omitting any reference to the random variable, is useful when the context is clear and we shall use it often. This omission is similar to saying, for example, "the probability of (H,T) is 1/4" instead of "the probability of getting (H,T) by tossing a fair coin twice is 1/4".

■ **Example 1.17** A fair coin is tossed twice. The probability space is given by $\Omega = \{T,H\}^2$. As this is a fair coin then $\forall \omega \in \Omega\ \mathcal{D}(\omega) = \frac{1}{4}$. Let X denote the number of heads obtained in each outcome (ω). The possible values of X are 0, 1, and 2. The probability of each value is

$$\mathcal{D}(x) = \begin{cases} \frac{1}{4} & x = 0,2 \\ \frac{1}{2} & x = 1 \\ 0 & \text{else} \end{cases}$$

■

Definition 1.3.7 Let Ω be a continuous probability space and X a random variable over Ω. We say that X is a *continuous random variable* if there exists $f(x) \geq 0$ so that we can write, for every $D \subset \mathbb{R}$

$$\mathcal{D}(X \in D) = \int_D f(x)\,dx$$

The function f is called the ***probability density function*** **(PDF)** of X.

In particular, this means that for any $a,b \in \mathbb{R}$, where $a \leq b$:

$$\mathcal{D}(X \in [a,b]) = \int_a^b f(x)\,dx$$

As before, f should actually be denoted by $f_X(x)$ to emphasize that it characterizes the random variable X, but often the subscript X is omitted. In this course we assume that $f(x)$ is a regular function and therefore the probability mass function for a single point is 0: $\mathcal{D}(X=a) = \int_a^a f(x)\,dx = 0$ for any $a \in \mathbb{R}$. Note that the probability density function satisfies that

- $f(x) \geq 0$
- $\int_{-\infty}^{\infty} f(x)\,dx = 1$
- $f(x)$ is a probability *density*, not a probability. For example, it could be that $f(x) > 1$.

1.3.1.3 Mean and Variance

When dealing with random variables (and later on with distributions) we are often interested in different measures of these random variables. The two most common and widely used measures are the mean (expected value) and variance.

Definition 1.3.8 The ***expected value*** of a random variable X is

$$\mathbb{E}[X] := \sum_{x \in \mathcal{X}} x\mathcal{D}(x) \qquad \text{or} \qquad \mathbb{E}[X] = \int_{-\infty}^{\infty} xf(x)\,dx$$

Claim 1.3.3 Let X,Y be two random variable with finite expected values $\mathbb{E}[X], \mathbb{E}[Y]$. Then:

- Linearity of expectation: $\mathbb{E}[aX+Y] = a\mathbb{E}[X] + \mathbb{E}[Y]$.
- Expectation of function of random variable (Law of the unconscious statistician): $\mathbb{E}[g(X)] = \sum g(x)\mathcal{D}(x)$.

- If X and Y are independent then: $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$.
- If $X \leq Y$ then $\mathbb{E}[X] \leq \mathbb{E}[Y]$.

Definition 1.3.9 Let X be a random variable with $\mathbb{E}[X]$, then the ***variance*** of X is

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

The ***standard deviation*** of X is defined as $\sigma := \sqrt{Var(X)}$.

Claim 1.3.4 Let X, Y be two random variables with finate expected values and variances. Then:

- $Var(X) = \mathbb{E}\left[X^2\right] - (\mathbb{E}[X])^2$
- For scalars $a, b \in \mathbb{R}$, $Var(aX + b) = a^2 Var(X)$
- The variance of X is non-negative: $Var(X) \geq 0$. Equality holds when $Var(X) = 0 \iff X$ is a constant.
- $$\begin{aligned} Var(X+Y) &= \mathbb{E}\left[(X + Y - \mathbb{E}(X) - \mathbb{E}(Y))^2\right] \\ &= Var(X) + Var(Y) + 2\mathbb{E}\left[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\right] \end{aligned}$$
- If $X_1, \ldots, X_n$ are independent then $V\left(\sum X_i\right) = \sum V(X_i)$

Definition 1.3.10 The ***covariance*** (joint variability) of X and Y is the expected value of the product of their deviations from the expected values

$$Cov(X,Y) := [(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$$

The covariance is also denoted as $\sigma(X,Y)$ or σ_{XY}.

Claim 1.3.5 Let X, Y be two random variables with finite expectation and variance. Then:

- Symmetry: $Cov(X,Y) = Cov(Y,X)$.
- For scalars $a, b, c, d \in \mathbb{R}$, $Cov(aX + b, cY + d) = a \cdot c \cdot Cov(X,Y)$
- $Cov(X,X) = Var(X)$

Note the different meaning implied by the above definitions: The variance measures the variation of a single random variable (like the height of a person in a population), whereas covariance is a measure of how much two random variables (like the height and weight) vary together.

■ **Example 1.18** Let Z be a random variable such that $P(Z = 1) = p, P(Z = 0) = 1 - p$. The variance of Z is: $Var[Z] = \mathbb{E}\left[Z^2\right] - (\mathbb{E}[Z])^2 = p - p^2 = p(1-p)$. ■

Exercise 1.9 Let $X \sim Unif([a,b])$ for $a, b \in \mathbb{R}$. Calculate the expectation and variance of X. ■

Proof.

$$\begin{aligned}\mathbb{E}[X] &= \frac{1}{b-a}\int_a^b x dx = \frac{b+a}{2} \\ \mathbb{E}\left[X^2\right] &= \frac{1}{b-a}\int_a^b x^2 dx = \frac{b^2+ab+a^2}{3}\end{aligned}$$

$$Var(X) = \frac{b^2+ab+a^2}{3} - \frac{(b+a)^2}{4} = \frac{(b-a)^2}{12}$$

■

1.3.2 A Little Statistics: Mean and Variance Estimation

The focus of this book is *Statistical* Machine Learning. When we study *Probability* we assume a known distribution and calculate probability of events of interest. When we study *Statistics* we turn the question on its head: using samples from an unknown probability distribution, we try to *estimate* or *test* properties of the unknown distribution. Recall the pizza slices example: we were interested in finding out how many pizza slices the participants will eat. We did not know what is the distribution of number of pizza slices participants eat. Given *samples* that were drawn from this unknown distribution, we attempted to *estimate* the distribution's mean. Let us formulate what we have done above and estimate both the expected value and the variance.

Let X be a random variable and let $\mathcal{D}(x)$ be its (unknown) distribution. Let $X_1,\ldots,X_m \overset{i.i.d}{\sim} D(x)$ be m random variables of this distribution. The *i.i.d* denotes these are *independent and identically distributed* random variables. We are given *data samples*: $x_1,\ldots,x_m$, where each x_i is a *number* obtained by sampling X_i. In statistics, $\mathbb{E}[X]$ is called the *population* mean - "population" added to emphasize that it is the mean of the unknown distribution, and not of the observed data. Similarly, $Var(X)$ is called the *population variance*.

There are many ways to estimate $\mathbb{E}[X]$ and $Var(X)$ based on those m samples, and each one is called an *estimator* (that is, the function that estimates the value of some parameter of a random variable). Throughout this book we are going to use different estimators for different tasks. For the case of sample mean and variance we will use:

- ***Sample mean*** (an estimator for the population mean)

$$\widehat{\mu}_X = \frac{1}{m}\sum_{i=1}^{m} x_i$$

- ***Sample variance*** (an estimator for the population variance)

$$\widehat{\sigma}_X^2 = \frac{1}{m-1}\sum_{i=1}^{m}(x_i - \widehat{\mu}_X)^2$$

As meantioned before, whenever the context is clear, we will omit the subscript X and write $\widehat{\mu}$ and $\widehat{\sigma}^2$ instead of $\widehat{\mu}_X$ and $\widehat{\sigma}_X^2$. Note that the sample variance is somewhat different from the variance of the x_i's because of the division by $m-1$ instead of m. The difference between the two will be explained later on in the book.

Exercise 1.10 Let X be some random variable. Show that the expected values of the samples mean and sample variance estimators equals to the true parameters they estimate. That is their expectation is the population mean and population variance.

Solution. Taking the expectation value of the sample mean one gets

$$\mathbb{E}(\widehat{\mu}_X) = \mathbb{E}\left(\frac{1}{m}\sum_{i=1}^{m} x_i\right) = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}(x_i) = \mathbb{E}(X)\frac{1}{m}\sum_{i=1}^{m} 1 = \mathbb{E}(X) = \mu_X$$

where we used the linearity property of the expectation and that the samples are $i.i.d$. Taking the expectation value of the sample variance one gets

$$\begin{aligned}\mathbb{E}\left(\widehat{\sigma}_X^2\right) &= \frac{1}{m-1}\sum_{i=1}^{m}\mathbb{E}\left((x_i-\widehat{\mu}_X)^2\right)\\ &= \frac{1}{m-1}\sum_{i=1}^{m}\mathbb{E}\left(x_i^2 - 2x_i\frac{1}{m}\sum_{j=1}^{m}x_j + \frac{1}{m^2}\sum_{k,j=1}^{m}x_k x_j\right)\end{aligned}$$

The X_i's are i.i.d which implies that $\mathbb{E}(x_i) = \mathbb{E}(X)$ and $\mathbb{E}\left(x_i^2\right) = \mathbb{E}\left(X^2\right)$ for every i, as well as that $\mathbb{E}(x_k x_j) = \mathbb{E}^2(X) = \mu_X^2$ for every $k \neq j$. Substituting into the above sums we obtain that:

$$\mathbb{E}\left(\widehat{\sigma}_X^2\right) = \mathbb{E}\left(X^2\right) - \mathbb{E}^2(X) = Var(X) = \sigma_X^2$$

■

Definition 1.3.11 Let $\widehat{\theta}$ be an estimator of θ. The *bias* of $\widehat{\theta}$ is the expected deviation between θ and the estimator: $B\left(\widehat{\theta}\right) := \mathbb{E}\left[\widehat{\theta}\right] - \theta$. $\widehat{\theta}$ is said to be *unbiased* if $B\left(\widehat{\theta}\right) = 0$.

From what we have seen in the above exercise, $\widehat{\mu}_X$ is an unbiased estimator of the population mean, μ_X, and $\widehat{\sigma}_X^2$ is an unbiased estimator of the population variance σ_X^2. If we calculate the expected variance of the x_i's using $\frac{1}{m}\sum_{i=1}^{m}(x_i - \hat{\mu}_X)^2$, we will see that we do not get the value σ_X^2, but rather is $\frac{m-1}{m}\sigma_X^2$. Therefore this estimator is a *biased* estimator of σ_X^2.

R Biased estimator, though there expectation is not the true value of the parameters can be useful in different cases. Therefore, we would sometimes use biased estimators and sometimes unbiased estimators.

1.3.3 Multivariate Probabilities

Up to now, we only dealt with random variables taking a single value. In many machine learning applications however, we often face a situation where we have multiple properties we would like to use in order to perform prediction. To do that we consider multivariate random variables and multivariate distributions.

Definition 1.3.12 — Random vector. A (column) ***random vector***: $X := (X_1, \ldots, X_d)^\top$, is a finite collection of random variables, denoted $X_1, \ldots, X_d$, defined on a common probability space $(\Omega, \mathcal{D})$.

Definition 1.3.13 — Joint distribution. Given random variables $X_1, \ldots, X_d$, that are defined on a probability space, the joint probability distribution for $X_1, \ldots, X_d$ is a probability distribution that gives the probability that each of $X_1, \ldots, X_d$ falls in any particular range (for continuous RVs) or discrete set (for discrete RVs) of values specified for that variable.

Definition 1.3.14 — Joint PDF of a random vector. Two random variables X_1 and X_2 are ***jointly continuous*** if there exists a non-negative function $f_{X_1,X_2} : \mathbb{R}^2 \to \mathbb{R}$, such that, for any set $A \in \mathbb{R}^2$, it holsd that

$$\mathcal{D}((X,Y) \in A) = \int_A f_{X_1,X_2}(x_1,x_2)\,dx_1 dx_2$$

The function f_{X_1,X_2} is called the ***joint probability density function (JPDF)*** of X_1 and X_2. Often, if the context is clear, one omits the subscripts of f and simply writes $f(x_1,x_2)$.

Given $X \coloneqq (X_1,\ldots,X_d)^\top$, each of the scalar random variables $X_1,\ldots,X_d$ can be characterized by its PDF. However, unless the scalar RV's are mutually independent, the PDF of each coordinate of a random vector does not completely describe the probabilistic behavior of the whole vector. For instance, the behavior of the random vectors $X = (X_1,X_2)$ and $\tilde{X} = (X_1,X_1)$ is drastically different even if X_1 and X_2 have an identical PDF. Consider for example, $X_1,X_2 \sim Unif([-a,a])$ and $X_2 = -X_1$ and compare the probability to find X in the square $[0,1]\times[0,1]$ to that of finding $\tilde{X}$ in the same square.

1.3.3.1 Normal Distribution

As an example of random vectors, we now consider specific family of distributions - the Multivariate Normal (also called Multivariate Gaussian) Distributions. Due to different limit theories, such as the Central Limit Theorem, and due to nice mathematical properties of this distribution, it is often used to model different probabilistic scenarios.

Definition 1.3.15 — (Univariate) Normal Distribution. A random variable x has a ***normal distribution*** with expectation μ and variance σ^2 if it has a PDF of the form:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp^{-\frac{1}{2\sigma^2}(x-\mu)^2}$$

In this case we write: $x \sim \mathcal{N}(\mu,\sigma^2)$

Definition 1.3.16 — Multivariate Normal Distribution. A random vector $\mathbf{x} \in \mathbb{R}^d$ has a ***multivariate normal distribution*** with expectation μ and covariance matrix (see definition below) Σ if it has a joint PDF of the form:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d\,|\Sigma|}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right\}$$

In this case we write: $\mathbf{x} \sim \mathcal{N}(\mu,\Sigma)$

Observe that definition 1.3.16 is generalization of 1.3.15, i.e. when $d = 1$ both definitions are same. Often, we are interested only in how one of the variates distributes, without the influence of the rest. For example, suppose we have joint probability function of the variables "height" and "weight" but we are interested in how the "height" distributes. To do so we would like to integrate out the "weight" variable. This leads to the following definition:

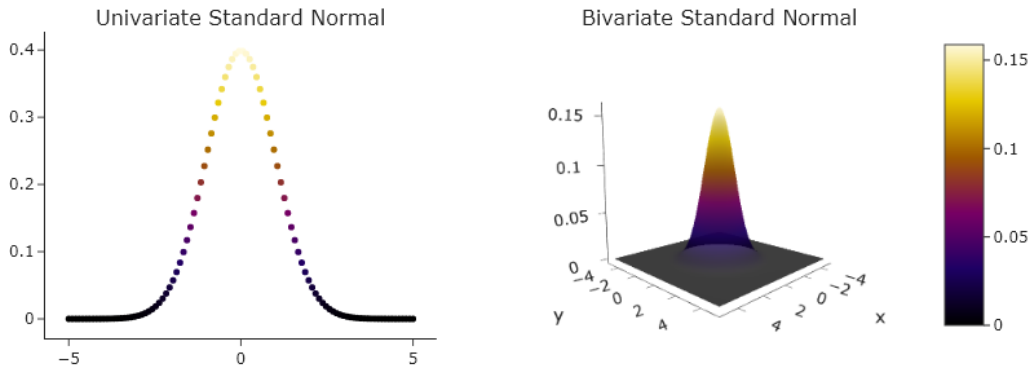


Figure 1.4: *Uni- and bivariate normal distributions.* *Chapter 1 Examples - Source Code*

Definition 1.3.17 The ***Marginal distribution*** of a subset if a collection of random variables with a joint probability distribution, is the probability distribution of the variables in the set:

$$f(\mathbf{x}) = \int_{\mathbf{y}} f(\mathbf{x},\mathbf{y})\, d\mathbf{y}$$

where the $\mathbf{y}$ integration is over all the random variables not in $\mathbf{x}$.

Exercise 1.11 — Marginal of Bivariate Normal. Let $\mathbf{x} \sim \mathcal{N}(\mu,\Sigma)$ where $\mathbf{x} \in \mathbb{R}^2, \mu = (\mu_1,\mu_2)^\top$ and $\Sigma = \begin{bmatrix} \sigma_1^2 & 0 \\ 0 & \sigma_2^2 \end{bmatrix}$ Find the PDF of the marginal distribution of x_1.

Observe that we can write the PDF as follows:

$$\begin{aligned}
f(\mathbf{x}) &= \frac{1}{\sqrt{(2\pi)^2|\Sigma|}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^\top \Sigma^{-1}(\mathbf{x}-\mu)\right) \\
&= \frac{1}{\sqrt{(2\pi)^2\sigma_1^2\sigma_2^2}} \exp\left(-\frac{1}{2}\begin{bmatrix} x_1-\mu_1 & x_2-\mu_2 \end{bmatrix}\begin{bmatrix} \sigma_1^{-2} & 0 \\ 0 & \sigma_2^{-2} \end{bmatrix}\begin{bmatrix} x_1-\mu_1 \\ x_2-\mu_2 \end{bmatrix}\right) \\
&= \frac{1}{\sqrt{(2\pi)^2\sigma_1^2\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2 - \frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right) \\
&= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right) \cdot \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right)
\end{aligned}$$

Using the definition of the marginal distribution:

$$\begin{aligned}
f(x_1) &= \int_{-\infty}^{\infty} f(x_1,x_2)\, dx_2 \\
&= \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right) \cdot \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right) dx_2 \\
&= \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right) \cdot \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp\left(-\frac{1}{2}\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2\right) dx_2
\end{aligned}$$

Notice that now we integrate a function of a uni-variate Gaussian for all values $x_2 \in (-\infty,+\infty)$.

Therefore this integral equals to 1 and we are left with:

$$f(x_1) = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp\left(-\frac{1}{2}\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2\right)$$

Which by definition 1.3.15 is a univariate Gaussian of the form $x_1 \sim \mathcal{N}\left(\mu_1, \sigma_1^2\right)$. ■

1.3.3.2 Covariance Matrix

Unlike the univariate case, when dealing with a multivariate distribution the variance is a matrix rather than a vector.

Definition 1.3.18 — Covariance Matrix. Let $\mathbf{X} \coloneqq (X_1, X_2, ..., X_d)^\top$ be a random vector. The *Covariance Matrix* Σ is the $d \times d$ matrix whose (i,j) entry is the covariance $\Sigma_{ij} \coloneqq \sigma(X_i, X_j)$:

$$\Sigma \coloneqq \begin{pmatrix} \mathbb{E}\left[(X_1 - \mathbb{E}[X_1])(X_1 - \mathbb{E}[X_1])\right] & \dots & \mathbb{E}\left[(X_1 - \mathbb{E}[X_1])(X_d - \mathbb{E}[X_d])\right] \\ \vdots & \ddots & \vdots \\ \mathbb{E}\left[(X_d - \mathbb{E}[X_d])(X_1 - \mathbb{E}[X_1])\right] & \dots & \mathbb{E}\left[(X_d - \mathbb{E}[X_d])(X_d - \mathbb{E}[X_d])\right] \end{pmatrix}$$

- In matrix notation we can express the covariance matrix as:

$$\Sigma \coloneqq \mathbb{E}\left[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\top\right]$$

- The diagonal elements of Σ are $\sigma(X_i, X_i) \equiv \sigma_{X_i}^2 \equiv Var(X_i)$. *Sigma* is a symmetric positive semi-definite matrix.

Back to statistics, let us consider how to estimate the covariance matrix based on a given sample. In this context, Σ is called the *population covariance matrix*. Let us start with the bi-variate case $d = 2$. Consider a two-dimensional random vector $\mathbf{X} = (X_1, X_2)^\top$, where, for example, X_1 is the height of a person and X_2 is the weight.

Taking a sample of m people from the population, we equip the data elements with two indices, the first stands for person's designated number and the second for the feature (1 for height, 2 for weight). Therefore, the height samples are denoted by $x_{1,1}, \dots, x_{1,m}$ and the corresponding weight samples are $x_{2,1}, \dots, x_{2,m}$. Note that we can write the data as a matrix $X \in \mathbb{R}^{m \times d}$ where m is the *sample size* and d is the dimension (number of random variables). In our case, $d = 2$ and

$$X = \begin{bmatrix} x_{1,1} & x_{2,1} \\ \vdots & \vdots \\ x_{m,1} & x_{m,2} \end{bmatrix} = (\mathbf{x}_1, \dots, \mathbf{x}_m)^\top$$

Definition 1.3.19 — Sample Covariance. The unbiased estimator of the ***sample covariance*** of the i'th and j'th random variables is given by:

$$\widehat{\sigma}(X_i, X_j) = \frac{1}{m-1} \sum_{k=1}^{m} (x_{k,i} - \widehat{\mu}_i)(x_{k,j} - \widehat{\mu}_j)$$

where $\widehat{\mu}_i$ is the sample mean of the random variable X_i.

In particular, notice that for the case where $i = j$ we are left with the unbiased estimator previously seen. We can now define the sample covariance matrix $\widehat{\Sigma}$.

Definition 1.3.20 — Sample Covariance Matrix. Let $X = (X_1, \ldots, X_d)^\top$ be a d-dimensional random vector. Let $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ be m i.i.d samples of X. The ***sample covariance matrix*** is a square d-by-d matrix $\widehat{\Sigma}$ such that $\widehat{\Sigma}_{i,j} = \widehat{\sigma}(X_i, X_j) \quad i, j = 1, \ldots, d$.

In matrix notation, the (biased) estimator for the sample covariance matrix is given by:

$$\widehat{\Sigma} := \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i - \widehat{\mu})(\mathbf{x}_i - \widehat{\mu})^\top = \frac{1}{m} \widetilde{\mathbf{X}}^\top \widetilde{\mathbf{X}}$$

for $\widetilde{\mathbf{X}}$ being the centered matrix: $\widetilde{\mathbf{X}}_{\cdot,i} := \mathbf{X}_{\cdot,i} - \widehat{\mu}$. The unbiased estimator is given by:

$$\widehat{\Sigma} := \frac{1}{m-1} \sum_{i=1}^{m} (\mathbf{x}_i - \widehat{\mu})(\mathbf{x}_i - \widehat{\mu})^\top = \frac{1}{m-1} \widetilde{\mathbf{X}}^\top \widetilde{\mathbf{X}}$$

Exercise 1.12 Let $\mathbf{X} = \begin{pmatrix} 150 & 45 \\ 170 & 74 \\ 184 & 79 \end{pmatrix}$ be samples of height and weight of 3 different people. Calculate the covariance matrix of the given sample.

Let us begin with centering the data. That is, subtract the empirical mean from each sample The sample mean is: $\widehat{\mu} = (168, 66)^\top$, so:

$$\mathbf{X}_{centered} = \mathbf{X} - \begin{pmatrix} 168 & 66 \\ 168 & 66 \\ 168 & 66 \end{pmatrix} = \begin{pmatrix} 150 & 45 \\ 170 & 74 \\ 184 & 79 \end{pmatrix} - \begin{pmatrix} 168 & 66 \\ 168 & 66 \\ 168 & 66 \end{pmatrix} = \begin{pmatrix} -18 & -21 \\ 2 & 8 \\ 16 & 13 \end{pmatrix}$$

Now, following the definition of the sample covariance matrix:

$$\widehat{\Sigma} = \frac{1}{3-1} \mathbf{X}_{centered}^\top \mathbf{X}_{centered} = \begin{pmatrix} 292 & 301 \\ 301 & 337 \end{pmatrix}$$

■

1.3.3.3 Linear Transformations of the Data Set

Let us look at how linear transformations affect the data set and as a result, the covariance matrix. Linear transformation can be of two types: scaling and rotating. Although we will now focus on the two-dimensional case, these results are easily generalized to higher dimensional data. The covariance matrix for two dimensions, $d = 2$, is

$$\Sigma = \begin{pmatrix} \sigma(X_1, X_1) & \sigma(X_1, X_2) \\ \sigma(X_2, X_1) & \sigma(X_2, X_2) \end{pmatrix}$$

We begin with a sample of points taken *i.i.d* from a bi-variate Gaussian with a zero mean and equal

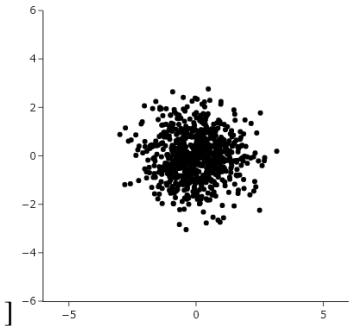


Figure 1.5: *Uncorrelated variables with identical variance. Chapter 1 Examples - Source Code*

variances $\sigma^2_{X_1} = \sigma^2_{X_2} \equiv \sigma^2$ (Figure 1.5). In particular this means that X_1 and X_2 are uncorrelated and the covariance matrix Σ is therefore of the form $\sigma^2 I_2$. In Figure 1.5 we can indeed see that there is no apparent relation between a points x coordinate and y coordinate. When we calculate the sample covariance matrix, as the sample size m increases, our estimation tends to $\widehat{\Sigma} = \sigma^2 I_2$.
Next, will us look at how linear transformations affect our data and the sample covariance matrix $\widehat{\Sigma}$ (Figure 1.6). We start transforming our data by multiplication by a scaling matrix:

$$S = \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix}$$

Notice how this transformation stretches (or shrinks) the x_1 and x_2 components of each sample by multiplying them by s_1 and s_2 respectively. In addition, as the scaling matrix is diagonal, the stretching of the x axis is uncorrelated with that of the y axis. The sample covariance matrix of the transformed data is:

$$\widehat{\Sigma}_{scaled} = \frac{1}{m-1}(\mathbf{X}S)^\top \mathbf{X}S = S^\top \left(\frac{1}{m-1}\mathbf{X}^\top \mathbf{X}\right) S = \begin{pmatrix} (s_1\widehat{\sigma})^2 & 0 \\ 0 & (s_2\widehat{\sigma})^2 \end{pmatrix}$$

.

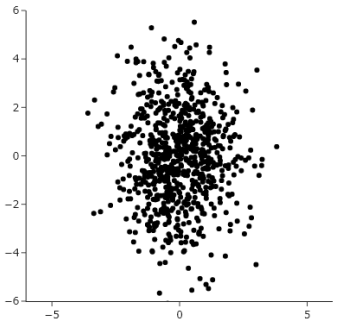


Figure 1.6: *Uncorrelated scaled variables with different scaling in the x and y axis. Chapter 1 Examples - Source Code*

Lastly, over the scaled data let us follow with a rotation transformation (Figure 1.7). Recall that any orthogonal matrix is in fact a rotation matrix, and specifically the rotation matrix of degree θ in $\mathbb{R}^2$

is given by:

$$R = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

As we can see in Figure 1.7 the transformed data shows a correlation between the x and y axes. To understand the reasoning behind let us calculate the sample covariance matrix of the scaled and then rotated data.

$$\begin{aligned} \widehat{\Sigma}_{rotated} &= = \tfrac{1}{m-1}(\mathbf{X}SR)^{\top}(\mathbf{X}SR) = R^{\top}S^{\top}\left(\tfrac{1}{m-1}\mathbf{X}^{\top}\mathbf{X}\right)SR = R^{\top}\left(S^{\top}\widehat{\Sigma}S\right)R \\ &= R^{\top}\begin{bmatrix} (s_1\widehat{\sigma})^2 & 0 \\ 0 & (s_2\widehat{\sigma})^2 \end{bmatrix}R = \widehat{\sigma}^2\begin{bmatrix} s_1^2\cos^2\theta + s_2^2\sin^2\theta & \sin\theta\cos\theta\left(s_2^2 - s_1^2\right) \\ \sin\theta\cos\theta\left(s_2^2 - s_1^2\right) & s_1^2\sin^2\theta + s_2^2\cos^2\theta \end{bmatrix} \end{aligned}$$

As the off diagonal element of the covariance matrix is non-zero, the two variables are correlated. Note that if we would not have applied an *asymmetric* scaling (i.e if $s_1 = s_2$), the off diagonal elements would have been zero. This means that rotation alone is not sufficient to induce correlations between random variables.

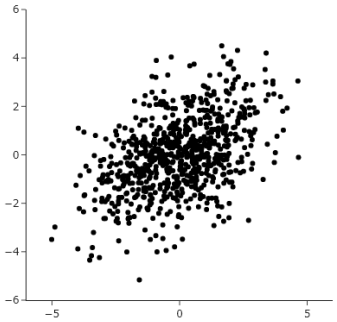


Figure 1.7: *Correlated variables produced by rotation of uncorrelated variables. Chapter 1 Examples - Source Code*

R The form obtained above, where we multiply an orthogonal matrix R by a diagonal matrix and then by another orthogonal matrix $R^{\top}$ is in fact the EVD of the sample covariance matrix of the transformed data. We will return to this point when we discuss the Principal Component Analysis algorithm (7.1.1).

1.3.4 Probability Inequalities

Being able to bound (from below or from above) the probability of different events is an important tool in machine learning. They are used in different senarios such as bounding the errors of a learning algorithm or concluding how good can we expect our algorithm to perform for a given sample size (and how this will change if we increase the sample size). For example, the law of large numbers states that if we take the empirical average (the sample mean) of enough i.i.d. random variables, it will be, most likely, very close to the expected value. Probability inequalities provide us a way to estimate how many samples are "enough".

■ **Example 1.19** Suppose we have a bag containing red and blue balls and we would like to estimate the fraction p of red balls in the bag by randomly picking one ball at a time and then putting it back in. The straightforward strategy is to draw m samples and then to use the following estimation ("prediction")

$$\widehat{p} = \frac{1}{m}\sum_{i=1}^{m} x_i = \frac{\text{Number of red balls}}{m}$$

where x_i equals 1 if the i'th ball came out red and 0 otherwise. It is clear that $\mathbb{E}[\widehat{p}] = p$. However, we might not reach this exact value due to the fact that we get only limited information. We are only sampling m balls so we might get "unlucky" and end up with a non-representative sample. Although less likely, this can happen even if m is very large, even larger than the total number of balls, because each time we returned the ball to the bag. Thus, typically, we must settle for a certain accuracy - an additive error - that is, we will be happy to know that if we sample more than m times, our estimation, $\widehat{p}$, is going to be within a distance $\varepsilon > 0$ from the real p. However, no matter how large m is, there is no absolute guarantee that $\widehat{p}$ will have that required accuracy.

Since m is finite, there is always a finite chance we sample the same color again and again. So, at most we can require to have enough confidence, say, with probability of $1-\delta$ where $\delta > 0$ is small, that $\hat{p}$ is accurate enough ("accurate enough" defined as within ε from our target, p). And now that we posed a realistic requirement, we can ask how large m should be in order to satisfy it. ■

The above example demonstrate the general type of questions we will use probability inequalities for: Given an accuracy parameter $\varepsilon > 0$ and a confidence parameter $\delta \in (0,1)$, how many samples are needed to guarantee that with probability of at least $1-\delta$, our estimate is within an additive error of at most ε. That is, we aim at constructing an algorithm (a "learner") whose estimations are probably (with probability at least $1-\delta$) approximately (within additive error of at most ε) correct. We then say that given m samples or more, our learner's estimation will be *probably approximately correct (PAC)*. The theoretical framework of PAC will be discussed in details in chapter 4.

1.3.4.1 Markov's and Chebyshev's inequalities

We now recall Markov's and Chebyshev's inequalities and then apply both inequalities to derive concentration inequalities for averages.

Theorem 1.3.6 Markov's inequality: Let X be a nonnegative random variable (i.e. $Im(X) \subseteq \mathbb{R}_{\geq 0}$) and deonte the expectation value of X by $\mathbb{E}[X]$. For any $a > 0$:

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a}$$

Proof. Let $f(x)$ be the density function of x. Since X is non-negative so $f(x) = 0$ for $x < 0$. Thus,

$$\frac{\mathbb{E}[X]}{a} = \frac{1}{a}\int_0^\infty f(x)x\,dx \geq \frac{1}{a}\int_{x=a}^\infty f(x)x\,dx \geq \frac{1}{a}\int_{x=a}^\infty f(x)a\,dx = \mathbb{P}(X \geq a)$$

■

Corollary 1.3.7 Let $X_1,\ldots,X_m$ be m i.i.d. non-negative random variables and denote the expectation value of each by $\mathbb{E}[X_i] = \mathbb{E}[X]$ $i = 1,\ldots,m$. Denote also $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$. Then for any

$a > 0$:

$$\mathbb{P}\left[\overline{X} \geq a\right] \leq \frac{\mathbb{E}[X]}{a}$$

Proof. As $\overline{X}$ is a non-negative random variable let us apply Markov's Inequality. So: $\mathbb{P}\left(\overline{X} \geq a\right) \leq \mathbb{E}\left[\overline{X}\right]/a$. Now

$$\mathbb{E}\left[\overline{X}\right] = \mathbb{E}\left[\frac{1}{m}\sum_{i=1}^{m} X_i\right] = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}[X_i] = \frac{1}{m}\sum_{i=1}^{m}\mathbb{E}[X] = \mathbb{E}[X]$$

and therefore, we obtain the desired bound. ■

Markov's inequality gives a bound in terms of the expectation value $\mathbb{E}[X]$, but it can be used to obtain also a bound in terms of the variance of X. This bound is called Chebyshev's inequality and does not require X to be non-negative.

Theorem 1.3.8 **Chebyshev's inequality:** For a random variable X with finite mean $\mathbb{E}[X]$ and a variance $Var(X)$ and for every $a > 0$

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq a] \leq \frac{Var(X)}{a^2}$$

Proof. Consider the random variable $Y = (X - \mathbb{E}[X])^2$. This is a non-negative random variable and as such we can apply Markov's inequality over it. We obtain that

$$\mathbb{P}\left[(X - \mathbb{E}[X])^2 \geq a^2\right] \leq \frac{Var(X)}{a^2}$$

To finish the proof, simply observe that $\mathbb{P}[|X - \mathbb{E}[X]| \geq a] = \mathbb{P}\left[(X - \mathbb{E}[X])^2 \geq a^2\right]$. ■

Unlike in the case of the expectation value, the variance of the average of i.i.d. random variables is not equal to the variance of the original random variable. For example, for m i.i.d. random variables, $X_1, \ldots, X_m$, with a variance $Var(X_i) = Var(X)$, $i = 1..m$, we have

$$V\left[\frac{1}{m}\sum_{i=1}^{m} X_i\right] = \frac{1}{m^2} V\left[\sum_{i=1}^{m} X_i\right] = \frac{1}{m^2}\sum_{i=1}^{m} Var(X_i) = \frac{1}{m^2}\sum_{i=1}^{m} Var(X) = \frac{1}{m} Var(X)\ .$$

where we used the fact the $Var[X_1 + X_2] = Var(X_1) + Var(X_2)$ for independent X_1 and X_2. In particular, the variance of the average tends to zero when m tends to infinity. This leads to a much better concentration inequality.

Corollary 1.3.9 Let $X_1, \ldots, X_m$ be m i.i.d random variables with a finite variance $Var(X)$. Denoting $\overline{X} = \frac{1}{m}\sum_{i=1}^{m} X_i$. For any $a > 0$, it holds that

$$\mathbb{P}\left[\left|\overline{X} - \mathbb{E}\left[\overline{X}\right]\right| \geq a\right] = \mathbb{P}\left[\left|\overline{X} - \mathbb{E}[X]\right| \geq a\right] \leq \frac{Var(X)}{m \cdot a^2}$$

For a positive integer k, the k-th *moment* of a random variable X is defined by $\mathbb{E}[X^k]$. As we just saw, Markov's inequality exploits only information about the first moment, while Chebyshev's inequality uses both the first and the second moments. Comparing the bounds in 1.3.7 and 1.3.9, we see that using both the first and the second moments, we obtain better concentration inequalities than using only the first. We shall see soon that more generally, the more moments we use, the better the bounds we get.

1.3.4.2 Coin Prediction Example

Let us consider the problem of estimating the bias of a coin, or in short 'coin prediction'. This will allow us to introduce the important concept of the *sample complexity*, to demonstrate the usefulness and limitations of the Markov's and Chebyshev's inequalities as well as a new one called Hoeffding's inequality.

Formally, a coin flip is a Bernoulli random variable Z which takes the value 1 for "Heads" or 0 for "tails". We shall denote the probability distribution of Z by $\mathcal{D}_p$, where $0 \leq p \leq 1$ and $\mathcal{D}_p(1) = p, \mathcal{D}_p(0) = 1-p$ are the probabilities to obtain 1 and 0 correspondingly. If $p = 1/2$ we say that the coin is fair. Let the following be a sequence of m tosses of this coin $Z_1, \ldots, Z_m \overset{i.i.d}{\sim} Ber(p)$ and denote the results of the tosses (that is, the samples) by $S = (z_1, \ldots, z_m)$. Denote the probability of obtaining a specific S by $\mathcal{D}_p^m(S)$ (or simply by $\mathcal{D}^m(S)$).

A coin prediction *learning algorithm*, $\mathcal{A}$, is a procedure which takes as an input a sequence S, drawn according to $\mathcal{D}_p^m$, and produces as an output an estimation of p. This estimation, also called "prediction", is denoted by $\mathcal{A}(S)$ or $\widehat{p}(S)$ or simply $\widehat{p}$. Since S is finite, we do not expect our estimation $\widehat{p}$ to be exact. Instead, we will settle for an algorithm that yields a $\widehat{p}$ which satisfies $|\widehat{p} - p|| \leq \varepsilon$, for some $0 < \varepsilon < 1$ which is called the *accuracy* parameter. Even then, there is always (unless p equals 1 or 0) *some* chance that the drawn sequence would be highly non-representative. For example, it can come out all 0's in spite of p being close to 1. So it is impossible to obtain a guarantee that $|\widehat{p} - p| \leq \varepsilon$ holds with absolute certainty.

Hence, we introduce a *confidence* parameter $\delta \in (0,1)$, and require that the event $\{S : |\widehat{p} - p| > \varepsilon\}$ occurs with a probability of at most δ. In other words, we require our algorithm to be such that the probability of flipping the coin m times and obtaining a sequence S that causes it to produce an inaccurate estimation, $\widehat{p}(S)$ ('inaccurate' meaning $|\widehat{p} - p| > \varepsilon$) is smaller or equal to δ.

Intuitively, the larger the number of flips, the more information we have about the coin and the better chance we have to satisfy the accuracy and confidence requirements. Since the accuracy and confidence parameters ε and δ are fixed, there should be some finite number of flips $m_{\mathcal{A}}$ (which depends on ε, δ and $\mathcal{A}$) such that for any sample of size $m \geq m_{\mathcal{A}}$ our algorithm satisfies the above accuracy and confidence requirements. If such $m_{\mathcal{A}}$ exists, we say that our algorithm is a *learning algorithm* for the task of coin prediction. Mathematically speaking, we are looking for an algorithm that satisfies the following definition:

Definition 1.3.21 Let $\mathcal{A}$ be an algorithm, which will be also denoted as $\widehat{p}(S)$ that given a set of coin tosses samples $S \in \{0,1\}^m$ returns $\widehat{p} \in [0,1]$ and satisfies the following conditions:

- For any $\varepsilon, \delta \in (0,1)$, there exists a non-negative integer $m_{\mathcal{A}}(\varepsilon, \delta)$ such that, if a sequence S of m numbers, where $m \geq m_{\mathcal{A}}$, is generated according to $\mathcal{D}_p^m$, then, **for any** $0 \leq p \leq 1$:

  $$\mathcal{D}_p^m\left[|\widehat{p}(S) - p| > \varepsilon\right] \leq \delta$$

  Namely, the probability to get a sample S such that the algorithm's output $\widehat{p}(S)$ will not be in the interval $[p \pm \varepsilon]$ is less or equals to δ.

- If a sequence S of m numbers, where $m < m_{\mathcal{A}}$, is generated, **there exists a** p, with $0 \leq p \leq 1$, such that:

  $$\mathcal{D}_p^m\left[|\widehat{p}(S) - p| > \varepsilon\right] > \delta$$

The function $m_{\mathcal{A}}(\varepsilon, \delta)$: $[0,1] \times [0,1] \to \mathbb{N}$ is called the *sample complexity* of the algorithm $\mathcal{A}$.

The first condition means that regardless to the true value of p, it is enough to draw $m_{\mathcal{A}}$ samples (i.e., to toss the coin $m_{\mathcal{A}}$ times) in order to know p, with a certainty of $1-\delta$ and an accuracy of $\pm\varepsilon$. The second conditions means that, at least for some values of p, drawing $m_{\mathcal{A}}(\varepsilon, \delta) - 1$ samples would *not* be enough for that.

Note that the confidence requirement must hold *independently of the procedure by which the coins are provided*. That is, independently of the way $\widehat{p}$ is chosen. It does not matter if, for example, the coin is randomly drawn from a pile of coins where most coins are approximately fair (most of their p's are close to $1/2$) or where most are faked in a specific way, (their p's are concentrated around, say, $\frac{1}{4}$), or where all p's are equally probable.

Choosing an Algorithm

Let us now look for an algorithm that satisfies the above definition. Given a sample $S = (z_1, \ldots, z_m)$ the most straightforward estimate of p is the empirical proportion of heads (ones), namely

$$\widehat{p}(S) = \frac{1}{m}\sum_{i=1}^{m} z_m$$

So we have a very simple algorithm for coin prediction, "Count the ones and divide by the number of tosses". Let us show that this algorithm satisfies the definition above.

We begin with noticing that this estimator, being the empirical mean of the parameter is an unbiased estimator of p. As such it can achieve, for the right sample S, that $|\widehat{p} - p| = 0$. Next, we will test the quality of $\widehat{p}$ as an estimator of p. In other words, how many coin flips we need to ensure that $\widehat{p}$ is, most probably, very close to its expectation value, p? To answer this question, we can use concentration inequalities.

Estimating Sample Complexity Using Markov's Inequality

For a direct application of Markov's inequality we take $|\widehat{p} - p|$ as our random variable. We then need to calculate $\frac{1}{\varepsilon}\mathbb{E}\left[|\widehat{p} - p|\right]$ and extract its dependence on the sample size m. By doing so we achieve the following bound:

$$\mathcal{D}_p^m\left[|\widehat{p} - p| \geq \varepsilon\right] \leq \frac{1}{\sqrt{4m\varepsilon^2}}$$

If we select m to be

$$m \geq \left\lceil \frac{1}{4\varepsilon^2} \cdot \frac{1}{\delta^2} \right\rceil$$

then the right-hand side is smaller or equals to δ. So, for any $\varepsilon, \delta \in (0,1)$, if we sample $m_{\mathcal{A}}(\varepsilon,\delta) \geq \left\lceil \frac{1}{4\varepsilon^2} \cdot \frac{1}{\delta^2} \right\rceil$ samples then this learning algorithm achieves that

$$\mathcal{D}_p^m \left[|\widehat{p}(S) - p| > \varepsilon \right] > \delta$$

Estimating Sample Complexity Using Chebyshev's Inequality

To improve the upper bound seen above (i.e find a sample complexity function that will need less samples), let us use Chebyshev's inequality. As the variance of a Bernoulli random variable is $p(1-p) \leq 1/4$, when applying corollary 1.3.9 we get:

$$\mathcal{D}_p^m \left[|\widehat{p} - p| \geq \varepsilon \right] = \mathcal{D}_p^m \left[|\widehat{p} - \mathbb{E}[\widehat{p}]| \geq \varepsilon \right] \leq \frac{p(1-p)}{m\varepsilon^2} \leq \frac{1}{4m\varepsilon^2}$$

We see that the bound obtained using Chebyshev's inequality tends to zero as $\frac{1}{m}$ while the one obtained from Markov's inequality tends to zero as $\frac{1}{\sqrt{m}}$. This fact enables us to obtain a better bound for the sample complexity:

Corollary 1.3.10 The sample complexity of coin prediction is bounded above by $m(\varepsilon,\delta) \leq \left\lceil \frac{1}{4\varepsilon^2} \cdot \frac{1}{\delta} \right\rceil$.

Estimating Sample Complexity Using Hoeffding's Inequality

A natural question which arises is whether the obtained bound is optimal (tight). Indeed, we can further improve the bound by exploiting the fact that our random variable not only has a finite variance, but it is also bounded between 0 and 1. For this end we use Hoeffding's inequality for the average of independent and bounded random variables.

Theorem 1.3.11 — Hoeffding's inequality. Let $X_1, \ldots X_m$ be independent and bounded random variables with $a_i \leq X_i \leq b_i$. Let $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$. Then,

$$\mathbb{P}\left[\left| \overline{X} - \mathbb{E}\left[\overline{X}\right] \right| \geq \varepsilon \right] \leq 2\exp\left(\frac{-2m^2\varepsilon^2}{\sum_{i=1}^m (b_i - a_i)^2} \right)$$

Corollary 1.3.12 Let $X_1, \ldots X_m$ be a sequence of m i.i.d random variables, each with an expectation value $\mathbb{E}[X]$ and all of which are bounded: $a \leq X_i \leq b$. Denote $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$ then:

$$\mathbb{P}\left[\left| \overline{X} - \mathbb{E}[X] \right| \geq \varepsilon \right] \leq 2\exp\left(\frac{-2m\varepsilon^2}{(b-a)^2} \right)$$

Proof. To conclude the corollary from Lemma 1.3.11 notice that as $X_1, \ldots, X_m$ all share the same expectation and bounds the:

$$\mathbb{E}\left[\overline{X}\right] = \frac{1}{m}\sum_{i=1}^m \mathbb{E}[X] = \mathbb{E}[X], \quad \sum_{i=1}^m (b_i - a_i)^2 = m \cdot (b-a)^2$$

By placing these expressions in Lemma 1.3.11 we conclude the inequality. ■

By applying Hoeffding's inequality to the case of coin prediction problem, then for a sample of size m, we obtain that:

$$\mathcal{D}_p^m \left[|\widehat{p} - p| \geq \varepsilon \right] \leq 2 \exp\left(-2m\varepsilon^2\right)$$

Therefore, using Hoeffding's Inequality we are able to get a bound which converges exponentially in m. By taking $m \geq \left\lceil \frac{1}{2\varepsilon^2} \cdot \log(\frac{2}{\delta}) \right\rceil$ samples we obtain that this probability is bound above by δ as required and conclude that:

Corollary 1.3.13 The coin prediction algorithm, $\widehat{p}(S)$, which estimates p by the number of ones (heads) divided by the number of coin flips, is a learning algorithm satisfying definition **??** with a sample complexity which is bounded above by $m_{\mathcal{A}}(\varepsilon, \delta) \leq \left\lceil \frac{1}{2\varepsilon^2} \cdot \log(\frac{2}{\delta}) \right\rceil$.

2. Linear Regression

2.1 Regression Models

Imagine that we work for an online store and would like to predict the "customer lifetime value", that is, the total future net revenue that an online customer will provide to the store. In order to do so, we choose different customer properties that we think might be relevant such as age, income, total spending in the website, average monthly visits, etc. The vector space defined over all possible values of these properties (commonly called features) is our **sample domain**. We denote it by $\mathcal{X}$. In the case of the online store $\mathcal{X} := \mathbb{R}^d$, for d the number of features. In addition, we also define the **response set** $\mathcal{Y}$ which in this case represents the customers' lifetime value. So in our case, $\mathcal{Y} = \mathbb{R}$.

Next, we collect these all these details for m customers. Each customer is represented as a *sample* – a pair $(\mathbf{x}, y)$ where $\mathbf{x} \in \mathcal{X}$ is the column vector of features of the sample (also called data point) and the corresponding response $y \in \mathcal{Y}$. This is our *training dataset*. We would like to use our training dataset to find a way to **estimate** or **predict** the lifetime values of any new customer using solely their feature vector. This setup is sometimes called **Batch Supervised Learning**. In order to do this, we will build a *regression model*.

A regression model is a way to represent a functional relation between a set of explanatory variables (features) in $\mathcal{X}$ and a scalar response, also referred to as dependent variable, in $\mathcal{Y}$. So, we will **assume** that there exists some function $f : \mathcal{X} \to \mathcal{Y}$ that captures this relation for each sample $x \in \mathcal{X}$ and its response $y \in \mathcal{Y}$. This function f is unknown to us and we would like to find it. It may be deterministic or it may contain a random component.

Let's assume first that the relation between $\mathbf{x} \in \mathcal{X}$ and $y \in \mathcal{Y}$ is *deterministic*. So we assume that there exists a function $f : \mathcal{X} \to \mathcal{Y}$ such that, each sample we observe, now or in the future, is of

the form $(\mathbf{x}, y)$ with $y = f(\mathbf{x})$. In particular for our training set $y_i = f(\mathbf{x}_i)$ for every training sample $i = 1 \ldots m$. Our goal is to *learn* f from a training sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, so we can estimate or predict the value $f(\mathbf{x})$ for a new value $\mathbf{x}$. A sample we haven't seen in our training set – a new sample – is sometimes called a *test* sample. Using the training sample S we will create a function that we hope is as similar as possible to the unknown function f. The function we create is called a **prediction rule** and we denote it by $\hat{f}$ or h_S. (The notation h_S emphasizes that our prediction rule depends on the training sample S).

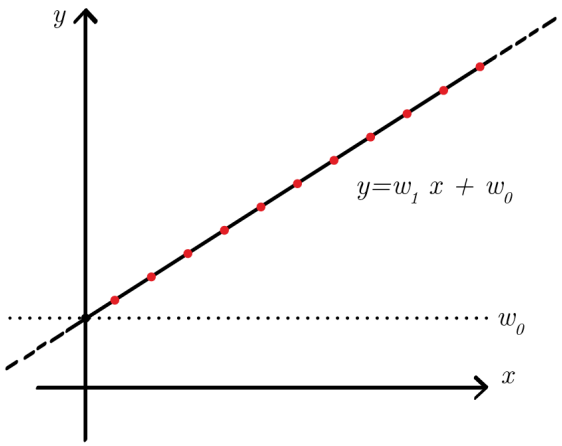


Figure 2.1: *Illustration of a regression model with $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$. Red dots are samples. The solid curve is the learned prediction rule $\hat{f}$.*

For reasons we discuss later (**??**), whenever we try to model a functional relation f, we restrict ourselves to a specific family of functions. Such a family is referred to as a *hypothesis class*. We decide on the hypothesis class before looking at the data, and the prediction rule we find must be in the chosen hypothesis class.

While we can build regression models over various domains $\mathcal{X}$, the simplest domain to consider is the Euclidean space $\mathbb{R}^d$ where each point x is a feature vector with d real numbers. In this chapter and in most of this book, we consider $\mathcal{X} := \mathbb{R}^d$.

2.1.1 Linear Regression

Let us assume that the relation $\mathcal{X} \to \mathcal{Y}$ is *linear*. This is perhaps the simplest relation we can describe. Formally, we define the *linear model*, or the *linear hypothesis class*, as the set of linear functions from the domain set to the response set:

$$\mathcal{H}_{reg} := \left\{ h(x_1, \ldots, x_d) = w_0 + \sum_{i=1}^{d} x_i w_i \;\middle|\; w_0, w_1, \ldots, w_d \in \mathbb{R} \right\} \tag{2.1}$$

In statistics, learning f from a training sample is known as *linear regression*[1]. Each function $h \in \mathcal{H}_{reg}$ is characterized by the **weights** (also known as regression coefficients) $w_1, \ldots, w_d$ representing the d

[1] The name "regression" refers to a statistical phenomenon known as "regression to the mean".

features and an **intercept** w_0. To simplify the notation, for a given sample $\mathbf{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$ we add a zero-th coordinate with the value of 1, and define $\mathbf{x} = (1, x_1, \ldots, x_d)^\top \in \mathbb{R}^{d+1}$. Using this notation each function in the linear hypothesis class can be written in the form $h(\mathbf{x}) := \langle \mathbf{x}, \mathbf{w} \rangle = \mathbf{x}^\top \mathbf{w}$. For the remainder of this chapter, we will assume that the intercept is already incorporated into the weights vectors, so we can define linear hypothesis class equivalently as

$$\mathcal{H}_{reg} := \left\{ h_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} \mid \mathbf{w} \in \mathbb{R}^{d+1} \right\} \tag{2.2}$$

Note that by convention, the first coordinate of $\mathbf{w}$ is the intercept w_0.

So, given a training set S, we are looking for a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ such that $y_i = \mathbf{x}_i^\top \mathbf{w}$ for all $i \in [m]$. This setup should be familiar from 1.1. However, as we will see, we may not be able to find a vector $\mathbf{w}$ for which all these equalities hold exactly.

The regression matrix

Let us arrange the training data $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ in matrix form. We define the *response vector* as the column vector $\mathbf{y} \in \mathbb{R}^m$ and the *regression matrix* (or *design matrix*) $\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$ as follows.

$$\mathbf{X} = \begin{bmatrix} — & \mathbf{x}_1 & — \\ — & \mathbf{x}_2 & — \\ & \vdots & \\ — & \mathbf{x}_m & — \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Note that m rows of $\mathbf{X}$ represent our m training samples and the $d+1$ columns of $\mathbf{X}$ represent the intercept and d features. In this notation, we are looking for a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ that satisfies a system of m linear equations in the variable $\mathbf{w}$,

$$\mathbf{X}\mathbf{w} = \mathbf{y} \tag{2.3}$$

We must have enough training samples

At this point, we will assume that $m \geq d+1$, namely, that we have enough training samples so that the linear system (2.3) is not under-determined. In practical terms, this means that we have at least as many training samples as we have features. In our online store example, this means that we must collect data on $m \geq d+1$ customers before we start training our regression model, where d is the number of features we collect on each customer (e.g. age, income, total spending, number of monthly visits to the website, etc).

2.1.2 Designing A Learning Algorithm

2.1.2.1 Realizablity

Recall that to derive the problem of finding $\mathbf{w} \in \mathbb{R}^{d+1}$ that satisfies (2.3) we have restricted ourselves to describing functional relations $\mathcal{X} \xrightarrow{f} \mathcal{Y}$ such that $f \in \mathcal{H}_{reg}$. The case where there exists a solution for (2.3) is called the **Realizable** case. Let $\hat{\mathbf{w}}$ be a solution for (2.3), then the prediction rule we choose is $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\mathbf{w}}$.

The case where there is no $f \in \mathcal{H}_{reg}$ that satisfies the system of equations (i.e there is no solution for the system) is called the **Non-Realizable** case. In this case, since we decided to choose a

prediction rule in $\mathcal{H}_{reg}$, we must settle for finding $\hat{f} \in \mathcal{H}_{reg}$ which is *"most fitting"* for our purposes.

Our learning algorithm for linear regression must address both the realizable and non-realizable cases. In the realizable case, to find the rule f, all we need to do is solve the linear system (2.3) for $\mathbf{w}$. But what will we do in the non-realizable case, where $f \notin \mathcal{H}_{reg}$? How should we choose the prediction rule $\hat{f}$?

2.1.2.2 Loss Function

One way to choose $\hat{f} \in \mathcal{H}_{reg}$ in the non-realizable case is to assign each $f \in \mathcal{H}_{reg}$ with some measure of quality, and choose the "best" f. The function defined to measure the quality is called a **loss function** and it measures the quality of the hypothesis by comparing between the true- and predicted values:

$$\sum_{i=1}^{m} L\left(f\left(\mathbf{x}_i\right), \hat{f}\left(\mathbf{x}_i\right)\right), \quad i = 1, \ldots, m$$

We will then pick the prediction rule which is "best fitting"/"most likely" given our training data and with respect to the loss function we chose. Two commonly used loss functions for regression problems are the **Absolute Value Loss**

$$L\left(y, \hat{f}\left(\mathbf{x}\right)\right) := \left|y - \hat{f}\left(\mathbf{x}\right)\right| \tag{2.4}$$

or the **Squared Loss**

$$L\left(y, \hat{f}\left(\mathbf{x}\right)\right) := \left(y - \hat{f}\left(\mathbf{x}\right)\right)^2 \tag{2.5}$$

We will focus on the linear regression setup when using the square loss function.

2.1.2.3 Empirical Risk Minimization

As we are concerned for the performance of a prediction rule $\hat{f}$ on a new data point $\mathbf{x}$ by $\left(\hat{f}\left(\mathbf{x}\right) - y\right)^2$, it makes sense to choose $\hat{f}$ that minimizes that same loss L *on the training data we already have*. This strategy for choosing $\hat{f}$ is known as **Empirical Risk Minimization**. For a given prediction rule $\hat{f} \in \mathcal{H}$, the quantity

$$\sum_{i=1}^{m} L\left(y_i, \hat{f}\left(\mathbf{x}_i\right)\right)$$

where $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ is our training data, is called the **empirical risk**. In the case of the square loss, the empirical risk of the linear function $\hat{f}\left(\mathbf{x}_i\right) = \mathbf{x}_i^\top \mathbf{w}$ is given by:

$$\sum_{i=1}^{m} \left(y_i - \mathbf{x}_i^\top \mathbf{w}\right)^2 = ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 = \left(\mathbf{y} - \mathbf{X}\mathbf{w}\right)^\top \left(\mathbf{y} - \mathbf{X}\mathbf{w}\right) \tag{2.6}$$

2.1.2.4 Least Squares

Minimizing the empirical risk of (2.6) means minimizing the sum of squares of the deviations of the responses from a linear function. In other words, we choose the linear function in $\mathcal{H}_{reg}$ that is closest to the responses in terms of the squared error distance. The deviation $y_i - \mathbf{x}_i^\top \mathbf{w}$ is called the i-th **residual** and the total empirical risk in our case is called **Residual Sum of Squares** (or **RSS**):

$$RSS_{\mathbf{X},\mathbf{y}}\left(\mathbf{w}\right) := ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2$$

To simplify notation we often write $RSS(\mathbf{w})$ keeping the dependence on $\mathbf{X}, \mathbf{y}$ implicit. So to learn the linear function by empirical Risk minimization we want to find

$$\underset{\mathbf{w}\in\mathbb{R}^d}{argmin}\, RSS(\mathbf{w}) = \underset{\mathbf{w}\in\mathbb{R}^d}{argmin}\, ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 \tag{2.7}$$

It is important to notice that the optimization problem (2.7) addresses both the realizable and non-realizable cases:

- In the realizable case, as $\mathbf{y} \in Im(\mathbf{X})$ we know there exists at least one solution $\widehat{\mathbf{w}}$ such that $\mathbf{X}\widehat{\mathbf{w}} = \mathbf{y}$. Such a solution will achieve a value of zero. As the RSS function is bounded below by zero, such a solution is therefore a minimizer of the RSS.
- In the non-realizable case, as $\mathbf{y} \notin Im(\mathbf{X})$ there is no solution $\widehat{\mathbf{w}}$ such that $\mathbf{X}\widehat{\mathbf{w}} = \mathbf{y}$. Therefore, no vector $\widehat{\mathbf{w}}$ will achieve a value of zero for the RSS objective. Instead, we decide to find a vector that is "good enough" in the sense of minimizing the squared loss.

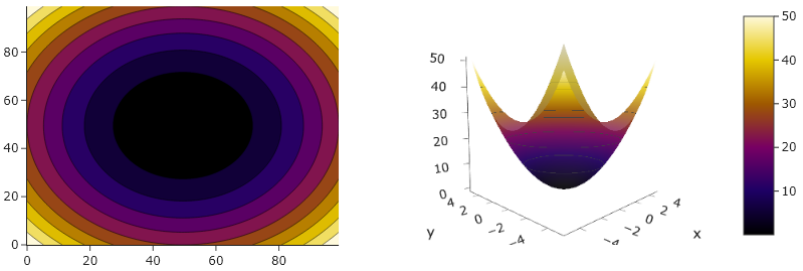


Figure 2.2: *Illustration of the RSS function over $\mathbb{R}^2$ for $\mathbf{X}$ of full rank. Chapter 2 Examples - Source Code*

A *necessary* condition for $\mathbf{w}$ to be a minimizer of the function $||\mathbf{y} - \mathbf{X}\mathbf{w}||^2$ is that all its partial derivative vanish at $\mathbf{w}$. Recalling the definition of the inner product, this condition can be written as:

$$\frac{\partial}{\partial w_j} RSS(\mathbf{w}) = -2\sum_{i=1}^{m} (\mathbf{x}_i)_j \cdot (y_i - \mathbf{x}_i\mathbf{w}) = 0 \tag{2.8}$$

for all $j = 0, \ldots, d$, where $(\mathbf{x}_i)_j$ is the j-th entry of $\mathbf{x}_i$. It is the $x_{j,i}$ element of the matrix $\mathbf{X}$. Notice that this constructs a system of $d+1$ linear equations in $\mathbf{w}$. We can organize (2.8) as such to get the form below. Recall that we have already derived this function in **??**.

$$\nabla RSS(\mathbf{w}) = -2\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0 \tag{2.9}$$

2.1.2.5 The Normal Equations

So a minimizer of (2.7) must also be a solution for the following linear system, known as the **Normal Equations**:

$$\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0 \quad \Longleftrightarrow \quad \mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\mathbf{X}\mathbf{w} \tag{2.10}$$

Geometric Interpretation

Let us derive a geometric interpretation of linear regression and gain a better understanding what the solution to (2.10) might be like. We usually think of $\mathbf{X} \in \mathbb{R}^{m\times(d+1)}$ as a matrix that consists of m rows, one for each training sample. Instead, we can equivalently think of $\mathbf{X}$ as a matrix that consists of $d+1$ columns, one for each feature (and the intercept). Define

$$\mathbf{X} \coloneqq \begin{bmatrix} | & & | \\ \varphi_0 & \cdots & \varphi_d \\ | & & | \end{bmatrix}$$

and recall that the vector space spanned by the columns of $\mathbf{X}$ is:

$$span\,(\varphi_0,\ldots,\varphi_d) = Im\,(\mathbf{X}) \subset \mathbb{R}^m$$

Since we assume $m \geq d+1$, $Im\,(\mathbf{X})$ is a linear subspace of $\mathbb{R}^m$. If we have many more samples than features, $m \gg d+1$, then $Im\,(\mathbf{X})$ is just a small subspace of $\mathbb{R}^m$. If we have the minimal number of samples possible, $m = d+1$, and the vectors $\varphi_0,\ldots,\varphi_d$ form an independent set, then the subspace fills the entire space: $Im\,(\mathbf{X}) = \mathbb{R}^m$.

Now, consider the response vector $\mathbf{y} \in \mathbb{R}^m$. Note that it may or may not belong to the subspace $Im\,(\mathbf{X})$:

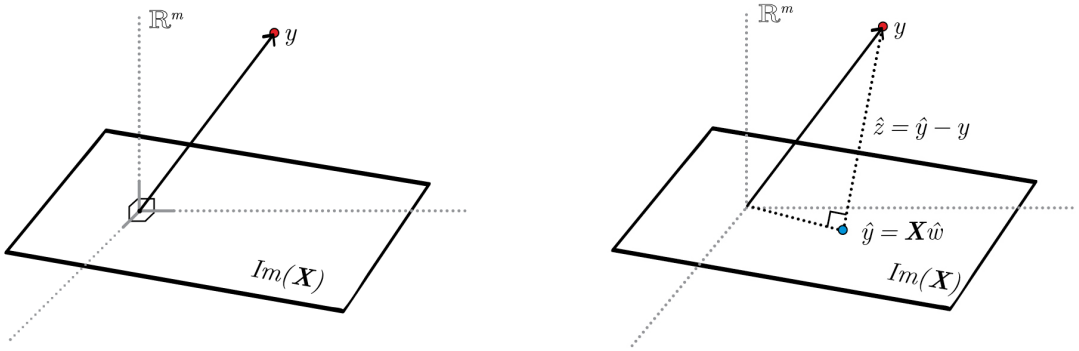
- If $\mathbf{y} \in Im\,(\mathbf{X})$ then by definition $\mathbf{y}$ is a linear combination of $\varphi_0,\ldots,\varphi_d$ and there exists a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ such that $\mathbf{X}\mathbf{w} = \mathbf{y}$. This is the realizable case. We can now differentiate between two sub-cases:
  - If $\varphi_0,\ldots,\varphi_d$ are linearly independent, then $\mathbf{y}$ can be expressed as a *unique* linear combination of the columns of $\mathbf{X}$. In this case the linear system (2.10) has a unique solution.
  - If however $\varphi_0,\ldots,\varphi_d$ are in fact linearly dependent, then there are infinitely many ways to express $\mathbf{y}$ as a linear combination of the columns of $\mathbf{X}$. Any one of these ways is a valid solution for (2.10).
- If $\mathbf{y} \notin Im\,(\mathbf{X})$ then $\mathbf{y}$ is not a linear combination of $\varphi_0,\ldots,\varphi_d$. As such there is no vector $\mathbf{w} \in \mathbb{R}^{d+1}$ that satisfies $\mathbf{X}\mathbf{w} = \mathbf{y}$. This is the non-realizable case. In this case we decided to choose the vector $\mathbf{w}$ for which $RSS(\mathbf{w}) = ||\mathbf{X}\hat{\mathbf{w}} - \mathbf{y}||$ is minimal.

Now we are able to understand what is "normal" about the normal equations (2.10). Observe that the equations (2.8), from which we have derived the normal equations, can be equivalently written as

$$\left\langle \varphi_j, \mathbf{y} - \mathbf{X}\mathbf{w} \right\rangle = 0 \quad j = 0,\ldots,d \tag{2.11}$$

We conclude that $\mathbf{w}$ is a solution to the normal equations if and only if $\mathbf{y} - \mathbf{X}\mathbf{w}$ is perpendicular to $\varphi_0,\ldots,\varphi_d$. Since these vectors span the subspace $Im\,(\mathbf{X})$, another way to write this is $\mathbf{y} - \mathbf{X}\mathbf{w} \in Im\,(\mathbf{X})^{\perp}$.

Let $\hat{\mathbf{w}}$ be a solution to the normal equations and define $\hat{\mathbf{y}} \coloneqq \mathbf{X}\hat{\mathbf{w}}$. Note that $\hat{\mathbf{y}}$, the vector where the i-th entry is the prediction on the i-th training sample $\mathbf{x}_i$, is $\hat{\mathbf{y}} \in Im\,(\mathbf{X})$. In this notation, when solving



(a) *In the non-realizable case, the response vector* $\mathbf{y}$ *lies outside* $Im(\mathbf{X})$*, the subspace spanned by the columns of* $\mathbf{X}$*. In this case there is no solution for the system* $\mathbf{Xw} = \mathbf{y}$*.*

(b) *If* $\hat{\mathbf{w}}$ *is a solution to the normal equations, then* $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ *is an orthogonal projection of the response vector* $\mathbf{y}$ *onto* $Im(\mathbf{X})$*. The difference* $\hat{\mathbf{z}} = \mathbf{y} - \hat{\mathbf{y}}$ *is therefore perpendicular (normal) to* $Im(\mathbf{X})$*.*

Figure 2.3: Geometric interpretation of linear regression

the normal equations, namely when seeking to minimize the RSS, we minimize $||\mathbf{y} - \hat{\mathbf{y}}||^2$. Define the **residual vector** $\hat{\mathbf{z}} := \mathbf{y} - \hat{\mathbf{y}}$. Note that from (2.11) we get that $\hat{\mathbf{z}} \in Im(\mathbf{X})^{\perp}$. In other words, if $\hat{w}$ is a solution to the normal equations then $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ is the **orthogonal projection** of $\mathbf{y}$ on $Im(\mathbf{X})$ and $\hat{\mathbf{z}} = \mathbf{y} - \hat{\mathbf{y}}$ is a *normal* (a perpendicular vector) to $Im(\mathbf{X})$. Hence the name the "normal equations".

Solving The Normal Equations

As we have seen, from a geometric perspective, if $m \geq d+1$, solving the normal equations means finding a vector $\hat{\mathbf{w}}$ such that $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ is the orthogonal projection of $\mathbf{y}$ on $Im(\mathbf{X})$. We can deduce from this two important facts about the existence and uniqueness of a solution to the normal equations:

- **Existence:** As a linear system, the normal equations can have either (i) no solutions, (ii) a unique solution, or (iii) an infinite number of solutions that constitute an affine subspace. From the geometric interpretation we see that (i) is impossible. Indeed it can be shown that the normal equations must have at least one solution, so that they have a unique solution or an infinite number of solutions.
- **Uniqueness:**
  - If the columns of $\mathbf{X}$ form a linearly independent set (equivalently, if $dim(Ker(\mathbf{X})) = 0$) then the projection $\hat{\mathbf{y}}$ can be described uniquely as a linear combination of the columns $\varphi_0, \ldots, \varphi_d$, namely, there exists a unique $\hat{\mathbf{w}}$ such that $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$. This vector of coefficients $\hat{w}$ is a unique solution to the normal equations.
  - If the columns of $\mathbf{X}$ contain linear dependencies (equivalently, if $dim(Ker(\mathbf{X})) > 0$) then the projection $\hat{\mathbf{y}}$ can be described as infinitely many linear combinations of the columns $\varphi_0, \ldots, \varphi_d$. Any such linear combination will suffice and we simply need to find *one* vector $\hat{w}$ such that $\hat{y} = \mathbf{X}\hat{\mathbf{w}}$.

Case 1: Linearly Independent Feature Vectors: $dim(Ker(\mathbf{X})) = 0$

It can be shown that for any matrix A then $Ker(A) = Ker(A^{\top}A)$. Since we assume that $Ker(\mathbf{X})$ is trivial, we know that the square symmetric matrix $\mathbf{X}^{\top}\mathbf{X}$ has a trivial kernel, namely, is invertible. Ex.1

This means that $\mathbf{w}$ satisfies $\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\mathbf{X}\mathbf{w}$ if and only if $\mathbf{w} = \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$. So in this case the unique solution to the normal equations is

$$\hat{\mathbf{w}} := \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$$

■ **Example 2.1** Let us find the estimator $\widehat{\mathbf{w}}$ for the following scenario. Suppose we are interested in estimating the running times in a 100 meters long race, besed on an athlete's height and weight. We gathered the details of the 4 top ranking athletes in the 2016 Rio Olympics:

Athlete	Weight (kg)	Height (cm)	Running Time (sec)
Usain Bolt	94	195	9.81
Justin Gatlin	79	185	9.89
Andre de Grasse	70	176	9.91
Yohan Blake	80	180	9.93

So the features are the *weight, height* and the response is *running time*. To fit a linear regression model to the data we begin with arranging it in a matrix and adding the intercept:

$$\mathbf{X} := \begin{bmatrix} 1 & 94 & 195 \\ 1 & 79 & 185 \\ 1 & 70 & 176 \\ 1 & 80 & 180 \end{bmatrix}, \quad \mathbf{y} := \begin{bmatrix} 9.81 \\ 9.89 \\ 9.91 \\ 9.93 \end{bmatrix}$$

As we have proven above, the estimator is given by the closed form of $\widehat{\mathbf{w}} := \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$. Over given data we obtain that $\widehat{\mathbf{w}} \approx (11.38, 0.003, -0.009)^\top$ (up to rounding up numbers).

Next, let us use this estimator to estimate the running times of a new sample $\mathbf{x} = (1, 74, 176)^\top$:

$$\widehat{y} = \mathbf{x}^\top\widehat{\mathbf{w}} = \left\langle \begin{bmatrix} 1 \\ 74 \\ 176 \end{bmatrix}, \begin{bmatrix} 11.38 \\ 0.003 \\ -0.009 \end{bmatrix} \right\rangle = 10.018$$

■

Case 2: Linearly Dependent Feature Vectors: $dim(Ker(\mathbf{X})) > 0$

The columns of $\mathbf{X}$ are linearly dependent and therefore there are infinitely many was to express the projection $\hat{\mathbf{y}}$ as a linear combination of the columns of $\mathbf{X}$. Since we need some way, it would be convenient if we could find a solution $\hat{w}$ that is close to the origin in $\mathbb{R}^{d+1}$ (rather than a solution with very large norm, say). An excellent way to do this uses the SVD of $\mathbf{X}$.

Definition 2.1.1 Let $\mathbf{X} \in \mathbb{R}^{m\times(d+1)}$ and let $\mathbf{X} = U\Sigma V^\top$ be its SVD. The **Moore-Penrose pseu-**

doinverse of $\mathbf{X}$ is $\mathbf{X}^\dagger = V\Sigma^\dagger U^\top$ where $\Sigma^\dagger$ is a $(d+1)\times m$ diagonal matrix defined by:

$$\Sigma^\dagger_{i,i} = \begin{cases} 1/\Sigma_{i,i} & \Sigma_{i,i} \neq 0 \\ 0 & \Sigma_{i,i} = 0 \end{cases}$$

This is a generalization of the inverse matrix and indeed when the matrix $\mathbf{X}$ is invertible then then $\mathbf{X}^\dagger = \mathbf{X}^{-1}$. An important property of the pseudoinvers is that for a linear system of equations $A\mathbf{x} = \mathbf{b}$ with an infinite number of solutions, then $A^\dagger\mathbf{b}$ is a solution with minimal ℓ_2 norm, namely Ex.2

$$A^\dagger\mathbf{b} = \text{argmin}\left\{||\mathbf{x}||_2 \;\middle|\; A\mathbf{x} = \mathbf{b}\right\} \tag{2.12}$$

In our case, a solution to the normal equations with minimal ℓ_2 norm, and as such closest to the origin with respect to the Euclidean norm, is given by

$$\hat{\mathbf{w}} := X^\dagger\mathbf{y}$$

It can be shown that when dealing with a matrix of linearly independent columns ($dim\,(Ker\,(\mathbf{X})) = 0$) then the previously found solution $\hat{\mathbf{w}} = \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$ equals to $X^\dagger\mathbf{y}$. We conclude that the formula $X^\dagger\mathbf{y}$ *always* gives us a solution to the normal equations: the unique solution if the solution is unique, and the solution with minimal ℓ_2 norm if not. Ex.3

It is left to show that the solution to the normal equations found above indeed minimize the RSS. We have derived the normal equations by asking that all partial derivatives of the RSS vanish. This means that a solution to the normal equations is an extremal point. Let us prove, for the case where $dim\,(Ker\,(\mathbf{X})) = 0$, that the found solution is indeed a minimum.

Claim 2.1.1 Assume $dim\,(Ker\,(\mathbf{X})) = 0$ and let $\hat{\mathbf{w}}$ satisfy $\mathbf{X}^\top\mathbf{y} = \mathbf{X}^\top\mathbf{X}\hat{\mathbf{w}}$. Then $\hat{\mathbf{w}}$ is a global minimizer of $RSS\,(\mathbf{w})$.

Proof. We know that $\hat{\mathbf{w}}$ is an extremal point of RSS. Now,

$$\frac{\partial^2 RSS_{\mathbf{X},\mathbf{y}}}{\partial w_k \partial w_l}\Big|_{\hat{\mathbf{w}}} = -2\frac{\partial \sum_{i=1}^m \left(y_i - \sum_{j=1}^d (\mathbf{x}_i)_j \hat{w}_j\right)\mathbf{X}_{\cdot,k}}{\partial w_l} = 2\sum_{i=1}^m \mathbf{X}_{\cdot,k}\mathbf{X}_{\cdot,l} = 2\left[\mathbf{X}^\top\mathbf{X}\right]_{kl} \quad \forall k,l \in [d+1]$$

The matrix $\mathbf{X}^\top\mathbf{X}$ is a positive semi-definite matrix, and since we assumed $dim(Ker(\mathbf{X})) = 0$, it strictly positive definite. It follows that $RSS(\hat{\mathbf{w}})$ is a minimum. ■

Corollary 2.1.2 The vector $\widehat{\mathbf{w}} := \mathbf{X}^\dagger\mathbf{y}$ is always a solution to the normal equations and a minimizer of optimization problem (2.7), namely, of the residual sum of squares (RSS).

2.1.3 Numerical Considerations When Implementing

So far we have designed the learning algorithm. Now we want to *implement* it, namely, write efficient code that implements the algorithm that we have designed. The field of *numerical linear algebra* assists in addressing this challenge as the implementation of every machine learning algorithm is eventually reduced to performing linear algebra computations (e.g. matrix-vector or matrix-matrix

products, matrix inverses and matrix decompositions). In the case of linear regression, as we have seen, to write software that trains a linear regression model, we need to be able to calculate a matrix SVD.

In your basic linear algebra courses you worked with mathematical objects over real and complex vector spaces. Likely, you did not stop to wonder how to compute (say) the inverse of a matrix on a computer. This is not as simple as it may sound. Computers do not calculate over $\mathbb{R}$, they use bits and more specifically floating-point arithmetics with finite precision. There is an entire field in the intersection of mathematics and computer science, known as numerical linear algebra, that studies the accuracy and complexity of algorithms for computing linear algebraic quantities and matrix decompositions. As a machine learning expert, you must be as knowledgable as possible regarding the numerical implementation of your learning algorithms. You should care *deeply* about how your algorithms are implemented and when they break numerically.

Let's see a simple example for a numerical consideration in our case of linear regression. (We will discover that the SVD is even more useful than we thought.) Recall that if $dim(Ker(\mathbf{X})) = 0$, and equivalently if $\mathbf{X}^\top\mathbf{X}$ is not singular (invertible) then we have a simple formula for training our linear regression model. But what happens if $\mathbf{X}^\top\mathbf{X}$ is "almost singular"?

Sometimes $\mathbf{X}^\top\mathbf{X}$ is formally invertible but *close to singular*. This happens if columns of $\mathbf{X}$ are almost co-linear or if one column of $\mathbf{X}$ is almost spanned by other columns. When this happens, if we are not careful we will run into numerical trouble. For example:

- Suppose we use the formula $\hat{\mathbf{w}} = \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$ and try to compute $[\mathbf{X}^\top\mathbf{X}]^{-1}$ using (say) Gauss elimination, we'll find that Gauss elimination may yield wildly incorrect results.
- Suppose we use the pseudoinverse formula and compute $\mathbf{X}^\dagger$. When $\mathbf{X}^\top\mathbf{X}$ is close to singular, we'll discover that the smallest singular values σ_i of $\mathbf{X}$ are very very small; when we try to compute $1/\sigma_i$ for the pseudoinverse with floating-point arithmetics, $1/\sigma_i$ will not be precise.

There is a simple practical solution for this problem: we choose a "numerical precision threshold" $\varepsilon > 0$ in advance. We can choose, say, $\varepsilon := 10^{-8}$. We then change the definition of the pseudoinverse slightly and define

$$\Sigma^{\dagger,\varepsilon}_{i,i} = \begin{cases} 1/\sigma_i & \sigma_i > \varepsilon \\ 0 & \sigma_i \leq \varepsilon \end{cases}.$$

This ensures that even if the columns of $\mathbf{X}$ are close to being linearly dependent our implementation will be numerically stable.

Recap

Before we continue, let's recap what we have seen so far:

- We have a regression problem over the sample space $\mathcal{X} = \mathbb{R}^d$ and response space $\mathcal{Y} = \mathbb{R}$. We have training data that consists of m samples, $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$, and m responses $y_1, \ldots, y_m \in \mathbb{R}$. The i-th coordinate of the vector $\mathbf{x}_i$ is the i features as measured for the i-th training sample. y_i is the response as measured for the i-th sample.
- We're looking for a linear prediction rule $\hat{f} \in \mathcal{H}_{reg}$. A linear prediction rule has an intercept w_0. To simplify notation, we create the m-by-$d+1$ regression matrix X (whose first column is a column of ones) and the response vector $\mathbf{y} \in \mathbb{R}^m$.
- Since any function in $\mathcal{H}_{reg}$ has the form $\mathbf{x} \mapsto \mathbf{x}^\top\mathbf{w}$ for some $\mathbf{w}$, we are looking to choose a

vector $\hat{\mathbf{w}} \in \mathbb{R}^{d+1}$

- We decided to choose the vector $\hat{w}$ that minimizes the function $RSS(\mathbf{w}) = ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2$. Why? in the realizable case where $\mathbf{y} = \mathbf{X}\mathbf{w}$ for some $\mathbf{w}$, the minimum of RSS is zero and the minimizer $\hat{\mathbf{w}}$ will satisfy $\mathbf{y} = \mathbf{X}\hat{\mathbf{w}}$. In the non-realizable case, where there is no solution to the system $\mathbf{y} = \mathbf{X}\mathbf{w}$, minimizing RSS will find a function in $\mathcal{H}_{reg}$ that best fits our training data (in the sense of empirical risk minimization for square loss).
- To find the vector $\hat{\mathbf{w}}$ that minimizes $RSS(\mathbf{w})$, we have to solve the normal equations. This is a linear system of equations that involves $\mathbf{X}$ and $\mathbf{y}$. To solve them, we calculate the singular value decomposition (SVD) of the regression matrix X and take $\hat{\mathbf{w}} = X^{\dagger}\mathbf{y}$. This works both when the columns of X (the training features) are linearly independent and when they are not. When they are independent, this recovers the famous linear regression formula $\hat{\mathbf{w}} = \left[\mathbf{X}^{\top}\mathbf{X}\right]^{-1}\mathbf{X}^{\top}\mathbf{y}$.
- There is a nice geometric interpretation of $\hat{w}$ as the expansion coefficients of $\hat{\mathbf{y}}$, the orthogonal projection of $\mathbf{y}$ on $Im(\mathbf{X})$, in the spanning set that consists of the features (the columns of X).
- When we find the vector $\hat{\mathbf{w}}$ this way, we say that we are *training a linear regression model* using our training data. Once we have found $\hat{\mathbf{w}}$, the training stage is over and we have a prediction rule: for a new (test) sample in $\mathbb{R}^d$ we add 1 to the vector and obtain $\mathbf{x} \in \mathbb{R}^{d+1}$, and predict its response using $\hat{f}(\mathbf{x}) = \mathbf{x}^{\top}\hat{\mathbf{w}}$.
- To make our learning algorithm numerically stable in the case where the columns of $\mathbf{X}$ are almost linearly dependent, we make a slight change in the definition of the pseudoinverse $\mathbf{X}^{\dagger}$ and only include the reciprocals of those singular values that are larger than a predetermined threshold ε.

2.1.4 A Statistical Model - Adding Noise

So far we assumed that the response y was a deterministic function of the sample $\mathbf{x}$, and that there was some deterministic function $f : \mathcal{X} \to \mathcal{Y}$ that underlies the relation $\mathcal{X} \to \mathcal{Y}$. This is an unrealistic assumption - in reality, measurements always contain randomness. In our online store example, we may consider that the revenue y measured for a customer is the sum of a deterministic component $\mathbf{x}^{\top}\mathbf{w}$ (where $\mathbf{x}$ is the customer's feature vector) and some random component z. This means that our dataset will not look like Figure 2.1 even if it is well described by the linear model. Instead is more likely to look like Figure 2.4.

To address this problem we describe a probabilistic model of the data. Let us assume, as before, that the relation $\mathcal{X} \to \mathcal{Y}$ linear, but with an additional factor capturing randomness in the relation. Suppose now that there exists a function $f : \mathcal{X} \to \mathcal{Y}$ such that the response for sample $\mathbf{x}$ is $y = f(\mathbf{x}) + z$ where z is some random variable. We assume that the noise z in a sample is identically distributed and independent of the noise in any other sample. In particular, our training sample S is

$$\left(x_i\,, f(\mathbf{x}_i) + z_i\right) \qquad i = 1, \ldots, m$$

with $z_1, \ldots z_m$ being *iid*: independent and identically distributed. Let us adapt the learning algorithm we designed for the deterministic case to the probabilistic (noisy) case. Let us choose the linear hypothesis class $\mathcal{H}_{reg}$ as before, so that our learning algorithm will output a linear prediction rule. We also assume that we have enough training data to learn, namely that $m \geq d + 1$. We assume that there is a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ such that for every sample vector $\mathbf{x}_i$ in our data follows the model:

$$y_i = \mathbf{x}_i^{\top}\mathbf{w} + z_i\,.$$

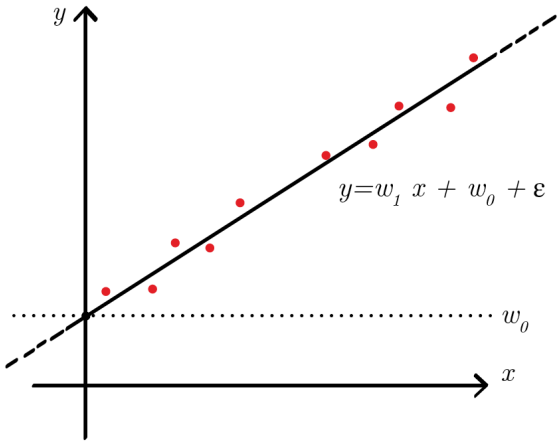


Figure 2.4: *Illustration of a linear regression model where training data is noisy*

Denoting the noise vector $\mathbf{z} \coloneqq (z_1, \ldots, z_m)^\top$ we have in matrix notation

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \mathbf{z}$$

Note that the vector $\mathbf{y}$ will typically not be in $Im(\mathbf{X})$, so that system $\mathbf{y} = \mathbf{X}\mathbf{w}$ has no solutions. As before, using the square loss function, and learning by the empirical risk minimization principal then:

$$\hat{\mathbf{w}} := \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{argmin} \; ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2$$

This means that our learning algorithm remains the same. We learn by solving the normal equation. As mentioned, $\mathbf{y} \notin Im(\mathbf{X})$ since the noise "pushed" $\mathbf{y}$ out of $Im(\mathbf{X})$. As we have seen, solving the normal equations is equivalent to projecting $\mathbf{y}$ back onto $Im(\mathbf{X})$, so our algorithms effectively attempts to remove the noise and recover the original prediction rule f.

2.1.4.1 The Maximum Likelihood principle

Another approach to solving the problem of linear regression with noise is as follows. Suppose we assume further that the noise is Gaussian $z_i \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$. This means that the i-th observation is independently distributed $y_i \sim \mathcal{N}(\mathbf{x}_i^\top \mathbf{w}, \sigma^2)$. In vector notation, we are assuming that the responses in our training sample follow a multivariate Gaussian distribution:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{X}\mathbf{w}, \sigma^2 I_m\right) \tag{2.13}$$

Now, suppose we *knew* the weight vector $\mathbf{w}$, we could then ask the following question: Given a fixed design matrix X and a known coefficients vector $\mathbf{w}$, what is the probability of observing the response vector $\mathbf{y}$? As each sample is independent to the others, the probability density is the product of the Gaussian densities of each sample

$$p(\mathbf{y}|\mathbf{w}) = \prod_{i=1}^{m} \left[\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\mathbf{x}_i^\top \mathbf{w} - y)^2}{2\sigma^2} \right) \right] \tag{2.14}$$

This is a question in probability: We know $\mathbf{w}$ and ask for the chance to observe $\mathbf{y}$?

However, when we design a learning algorithm, we are actually interested in the reverse question. We have the training sample, including the response vector $\mathbf{y}$. We are interested in a way to choose a linear prediction rule in $\mathcal{H}_{reg}$ and, equivalently, a vector $\mathbf{w}$. We can ask: what is the most "likely" value of $\mathbf{w}$ given the response vector that we observed. This approach is known as the **Maximum Likelihood** (ML) principle. This principle suggests that we choose $\mathbf{w}$ for which the probability density of getting the observed $\mathbf{y}$ is maximal. To make this formal, we first define the likelihood function:

Definition 2.1.2 — Likelihood. Let X be a random variable following some probability distribution $\mathcal{F}$ with a density function f that depends on a parameter $\theta \in \Theta$. The *likelihood function* is

$$\mathcal{L}(\theta|X) := f_\theta(X).$$

As an example, let's find the likelihood function in our case – as a function of the vector $\mathbf{w}$ – given an observed response vector $\mathbf{y}$:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}|X,\mathbf{y}) &= \mathbb{P}\left(y_1 = \mathbf{x}_1^\top \mathbf{w}, \ldots, y_m = \mathbf{x}_m^\top \mathbf{w} | \mathbf{X}, \mathbf{w}\right) \\
&= \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2}{2\sigma^2}\right) \\
&= \frac{1}{(2\pi\sigma^2)^{m/2}} \prod_{i=1}^m \exp\left(-\frac{\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2}{2\sigma^2}\right) \\
&= \frac{1}{(2\pi\sigma^2)^{m/2}} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^m \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2\right)
\end{aligned}$$

Formally, the maximum likelihood estimator chooses the parameter that maximizes the likelihood function:

Definition 2.1.3 — Maximum Likelihood Estimator. Let $\mathcal{L}$ be the likelihood function of some probability distribution $\mathcal{F}$ depending on parameter $\theta \in \Theta$ and let X a random variable following $\mathcal{F}$ with parameter θ. The *Maximum Likelihood Estimator* (MLE) for θ is

$$\widehat{\theta}^{MLE} := \underset{\theta \in \Theta}{argmax}\, \mathcal{L}(\theta|X)$$

As an example, let us find the MLE for our linear regression model **??**:

$$\begin{aligned}
\widehat{\mathbf{w}}^{MLE} &= \underset{\mathbf{w}}{argmax}\, \mathcal{L}(\mathbf{w}|\mathbf{y}) \\
&= \underset{\mathbf{w}}{argmax}\, \log \mathcal{L}(\mathbf{w}|\mathbf{y}) \\
&= \underset{\mathbf{w}}{argmax}\, \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^m \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2\right) \\
&= \underset{\mathbf{w}}{argmin}\, \sum_{i=1}^m \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2
\end{aligned}$$

we therefore conclude that the MLE (assuming i.i.d Gaussian noise) is identical to the Least Squares estimator obtained from a completely different principle - that of empirical risk minimization.

2.2 Polynomial fitting

We now turn to a specific example of linear regression, which will help us gain important general insights, and is also quite useful in and of itself. We will use our learning algorithm for linear

regression to predict the value of an unknown real function $g : \mathbb{R} \to \mathbb{R}$. We are given points $a_1 < a_2 < \ldots < a_m \in \mathbb{R}$ and labels $y_1, \ldots, y_m$. In the noiseless case, we know that $y_i = g(a_i)$ for an unknown g. We would like to predict the value of g on points beyond the training set. The hypothesis class is:

$$\mathcal{H}^d_{poly} = \left\{ x \mapsto p_{\mathbf{w}}(x) \,|\, \mathbf{w} \in \mathbb{R}^{d+1} \right\} \tag{2.15}$$

where $p_{\mathbf{w}}(a) = \sum_{i=0}^{d} w_i a^i$.

Given a training sample $\{(a_i, y_i)\}_{i=1}^m$ we would like to choose the polynomial coefficient vector $\mathbf{w}$ using least squares, namely to find

$$\hat{w} := \underset{\mathbf{w} \in \mathbb{R}^{d+1}}{argmin} \frac{1}{m} (y_i - p_{\mathbf{w}}(a_i))^2 \tag{2.16}$$

Let's create an equivalent linear regression problem. Define the design matrix $\mathbf{X}$

$$\mathbf{X} = \begin{bmatrix} 1 & a_1 & a_1^2 & \cdots & a_1^d \\ 1 & a_2 & a_2^2 & \cdots & a_2^d \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & a_m & a_m^2 & \cdots & a_m^d \end{bmatrix}$$

Notice that this is a *Vandermonde matrix*, hence it is of full rank. Convince yourself that using our learning algorithm for linear regression for the problem defined by $\mathbf{X}$ and $\mathbf{y}$ finds the vector $\hat{\mathbf{w}}$ from (2.16). After finding $\hat{\mathbf{w}}$, we can predict the value of the unknown function g at a new point a using the value $p_{\hat{\mathbf{w}}}(a)$.

R Here we discuss polynomial fitting where $\mathcal{X} = \mathbb{R}$. With very little adaptation, we could also allow the input data to be $\mathcal{X} = \mathbb{R}^d$, $d > 1$. In such cases the defined polynomial could include terms of multiplication of two (or more) features. We will encounter such an example in 8.

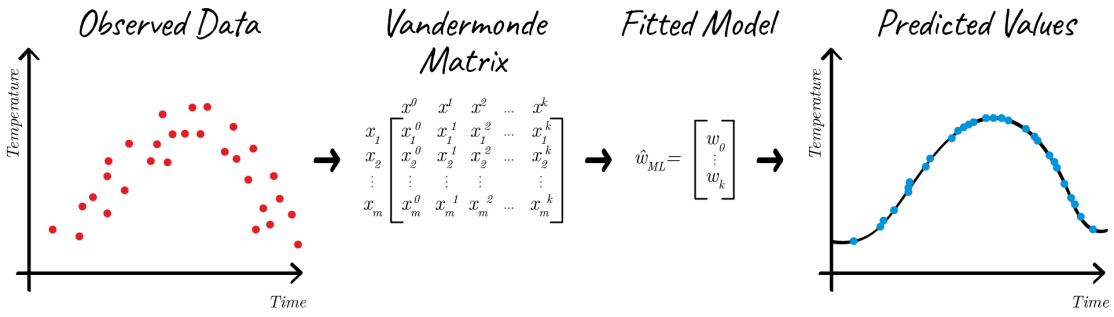


Figure 2.5: Scheme of Polynomial Fitting: *Dataset of hourly temperature where x_i denotes time of day and y_i the temperature. Fitting polynomial of degree* 4.

2.2.1 Bias and Variance of Estimators

Given a regression problem $\mathbf{X},\mathbf{y}$ we have seen how to solve $\mathbf{y}=\mathbf{X}\mathbf{w}+\varepsilon$. That is, finding a vector $\mathbf{w}$ that satisfies: $\mathbf{y}\approx\mathbf{X}\mathbf{w}$. We showed that the vector minimizing the sum of square distances is given by $\hat{\mathbf{w}}=\mathbf{X}^{\dagger}\mathbf{y}$. It is important to notice that as $\mathbf{y}$ is a random variable, the least squares estimator is a random variable. Therefore, we could look at different properties of such estimator. Specifically, we will look at the *bias* and *variance* of an estimator.

Definition 2.2.1 Let $\hat{\theta}$ be an estimator of θ. The *bias* of $\hat{\theta}$ is the expected deviation between θ and the estimator: $B\left(\hat{\theta}\right):=\mathbb{E}\left[\hat{\theta}\right]-\theta$. $\hat{\theta}$ is said to be *unbiased* if $B\left(\hat{\theta}\right)=0$.

Definition 2.2.2 Let $\hat{\theta}$ be an estimator of θ. The variance of $\hat{\theta}$ is the expected value of the squared sampling deviations: $var\left(\hat{\theta}\right):=\mathbb{E}\left[\left(\hat{\theta}-\mathbb{E}\left[\hat{\theta}\right]\right)^2\right]$.

What is the expectation in both the bias (2.2.1) and the variance (2.2.2) been calculated over? An estimator is a function over a sample $S=x_1,\ldots,x_m\in\mathbb{R}^d$ used to estimate some parameter: $\hat{\theta}\left(x_1,\ldots,x_m\right)\overset{?}{\approx}\theta$. As such, the expectation of $\hat{\theta}$ is over the selection of the samples. Going back to the least squares estimator, we could ask what is it's bias and variance.

Claim 2.2.1 Let $\mathbf{X},\mathbf{y}$ be a regression problem such that $\mathbf{y}=\mathbf{X}\mathbf{w}+\varepsilon$, $\varepsilon\sim\mathcal{N}\left(0,\sigma^2 I_d\right)$ and $\hat{\mathbf{w}}$ being the least squares estimator. Show that $\hat{\mathbf{w}}$ is an unbiased estimator.

Proof.

$$\begin{aligned}\mathbb{E}\left[\hat{\mathbf{w}}\right] &= \mathbb{E}\left[\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}\right]\\ &= \mathbb{E}\left[\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\left(\mathbf{X}\mathbf{w}+\varepsilon\right)\right]\\ &= \mathbb{E}\left[\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{X}\mathbf{w}\right]+\mathbb{E}\left[\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\varepsilon\right]\\ &= \mathbb{E}\left[\mathbf{w}\right]+\left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbb{E}\left[\varepsilon\right]=\mathbf{w}\end{aligned}$$

where the last equality is because $\mathbb{E}\left[\varepsilon\right]=0$ and $\mathbf{w}\in\mathbb{R}^d$. ■

To get some intuition about these two properties, let us revisit polynomial fitting. Consider for example the polynomial

$$Y=X^4-2X^3-0.5X^2+1+\varepsilon\quad\varepsilon\sim\mathcal{N}(0,2)\tag{2.17}$$

and let $x_1,\ldots,x_m$ be a set of samples where $x_i\in[-2,2]$. For these observations let us create 10 different datasets, generated according to the model above. For each dataset we use $x_1,\ldots,x_m$ and generate the response value $y_1,\ldots,y_m$ with the addition of the noise. Figure Figure 2.6 shows the different datasets generated by the model above and the fitted polynomial of degree 1. Black, red and blue points represent the true model, the observed data-points (with the sample noise) and the fitted model over the observed data-points. Notice how the different datasets yield different predicted models. This is the randomness of the prediction, driven by the randomness of the trainset. Over

these datasets we can now ask, for each value of x, what is the average prediction and its variance:

- In green is the average prediction of y for a given x across all datasets. The difference between the green and black lines capture the concept of the bias.
- In grey is the area of $\mathbb{E}[\hat{y}] \pm 2 \cdot Var(\hat{y})$ for a given x, also known as the confidence interval. The wider this area is, the more out prediction of $\hat{y}$ varies for different samples. This area captures the concept of the variance.

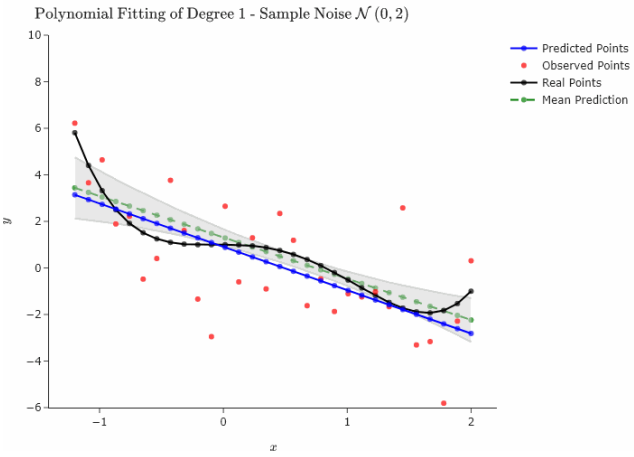


Figure 2.6: Polynomial Fitting: *Fitted polynomial of degree* 1 *over different datasets differing only in values of added sample noise.* *Chapter 2 Examples - Source Code*

Two phenomena are visible. The first is that the average distance of the fitted model (in green) and the true model (in black) is large. This means that our hypothesis class doesn't have sufficient expressive power to learn the true model. As such, we conclude that the **bias** of our estimator is high. The second is that the fitted models over different datasets do not differ by much. As such, we conclude that the **variance** of our estimator is low.

Next, consider the same setup as before but with the fitting of a polynomial of degree 8 (Figure 2.7). This time the different between the average prediction at each x and the true value of x is lower, while the differences between the fitted models (as indicated by the confidence intervals) is much higher. So the **bias** is low and the **variance** is high. As we enable more "flexible" (i.e. complex) models we are able to fit a model better to our given sample. However, as seen in Figure 2.7, if the model is too complex we might actually be fitting a model to the noise, rather than the actual true signal.

R It is important to note that what is seen in the figures are not the bias and variance of $\widehat{\mathbf{w}}^{LS}$ themselves but how these manifest over the shown datasets.

Interestingly, these two properties of bias and variance are linked. Let $\widehat{\mathbf{y}} = \widehat{\mathbf{y}}(S)$ denote the estimator of $\mathbf{y}$ when using $\widehat{\mathbf{w}}^{LS}$, and $\mathbf{y}^*$ the true $\mathbf{y}$ values. When solving the regression problem we wanted to minimize the mean square error between $\widehat{\mathbf{y}}$ and $\mathbf{y}^*$. What would be the expected MSE value?

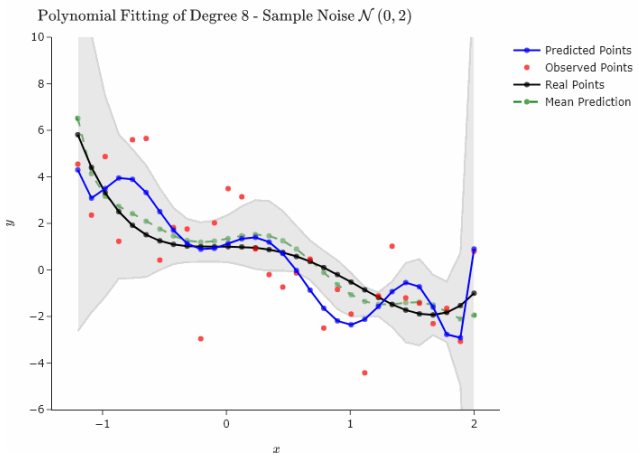


Figure 2.7: Polynomial Fitting: *Fitted polynomial of degree* 8 *over different datasets differing only in values of added sample noise.* *Chapter 2 Examples - Source Code*

Denote $\overline{\mathbf{y}} = \mathbb{E}[\widehat{\mathbf{y}}]$ so:

$$\begin{aligned}
\mathbb{E}\left[(\widehat{\mathbf{y}} - \mathbf{y}^*)^2\right] &= \mathbb{E}\left[(\widehat{\mathbf{y}} - \overline{\mathbf{y}} + \overline{\mathbf{y}} - \mathbf{y}^*)^2\right] \\
&= \mathbb{E}\left[(\widehat{\mathbf{y}} - \overline{\mathbf{y}})^2\right] + 2(\widehat{\mathbf{y}} - \overline{\mathbf{y}})\,\mathbb{E}[\widehat{\mathbf{y}} - \overline{\mathbf{y}}] + (\overline{\mathbf{y}} - \mathbf{y}^*)^2 \\
&= \mathbb{E}\left[(\widehat{\mathbf{y}} - \overline{\mathbf{y}})^2\right] + (\overline{\mathbf{y}} - \mathbf{y}^*)^2 \\
&= var(\widehat{\mathbf{y}}) + Bias^2(\widehat{\mathbf{y}})^2
\end{aligned} \tag{2.18}$$

Namely, we could **decompose** the generalization error (expected square loss between prediction and true value) into a variance component and a (squared) bias component.

$$MSE(\widehat{\mathbf{y}}) = Var(\widehat{\mathbf{y}}) + Bias(\widehat{\mathbf{y}}) \tag{2.19}$$

This means, that whenever we devise some estimator over our training data, the generalization error is influenced by both these factors. This is called the **Bias-Variance Trade-off**.

2.3 Summary and Exercises

1. Let $\mathbf{A}$ be some matrix. Show that $Ker(\mathbf{A}) = Ker(\mathbf{A}^\top \mathbf{A})$.
2. Let $\mathbf{A}$ be an invertible matrix. Show that $\mathbf{A}^\dagger = \mathbf{A}^{-1}$.
3. Let $\mathbf{X}, \mathbf{y}$ be a linear regression problem where the columns of $\mathbf{X}$ are linearly independent. Show that $\left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top = \mathbf{X}^{\dagger\top} \mathbf{y}$.

3. Classification

3.1 Classification Overview

In the previous chapter we discussed learning a regression problem where the response is a continuous value $\mathcal{Y} = \mathbb{R}$. When the response set $\mathcal{Y}$ is a finite set, this is a **classification** problem. We distinguish between classification problems where $|\mathcal{Y}| = 2$ (such as $\mathcal{Y} = \{\pm 1\}$ or $\mathcal{Y} = \{0, 1\}$) and multi-classification problems where $\mathcal{Y} = \{1, \ldots, k\}$. In the binary classification problem (or just "classification"), we provide a "yes"/"no" prediction. In a multi-class classification, we predict one of $k > 2$ classes. For most, we restrict out discussion only to binary classification problems, though all methods below can be generalized to k classes. Also, we will only deal with the Euclidean sample space $\mathcal{X} = \mathbb{R}^d$, namely, each sample has d **features**. Therefore out setup is as follows:

$$\mathcal{X} := \mathbb{R}^d, \mathcal{Y} := \{\pm 1\} \tag{3.1}$$

Classification Problems Examples

- Predict whether a patient will develop a certain medical condition, or not.
- Predict whether a user will like a new product, or not.
- Determine if a given network traffic pattern is one of a cyber attack or not.
- Determine whether an art work is an original or forged.
- Determine whether a given email is spam or not.
- Detect fraud on credit card transactions.
- Predict whether a loan applicant will default on the loan.
- (Multi-class) What are the objects seen in a given picture.

■ **Example 3.1** Seen in Figure 3.1 are samples of the "South Africa Heart Disease" dataset. Given the parameters of blood pressure, smoking, family history, etc., could we predict who has/will have coronary heart disease (chd)? Notice that some of the features are numerical (e.g. tabacco, ldl, etc.) while some are categorical (e.g famhist). ■

	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
0	160	12.00	5.73	23.11	Present	49	25.30	97.20	52	1
1	144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
2	118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
3	170	7.50	6.41	38.03	Present	51	31.99	24.26	58	1
4	134	13.60	3.50	27.78	Present	60	25.99	57.34	49	1
...	...	...	...	...	...	...	...	...	...	...
457	214	0.40	5.98	31.72	Absent	64	28.45	0.00	58	0
458	182	4.20	4.41	32.10	Absent	52	28.61	18.72	52	1
459	108	3.00	1.59	15.23	Absent	40	20.09	26.64	55	0
460	118	5.40	11.61	30.79	Absent	64	27.35	23.97	40	0
461	132	0.00	4.82	33.41	Present	62	14.70	0.00	46	1

Figure 3.1: Example classification dataset: *South African Heart Data from Elements of Statistical Learning*

Visualizing The Feature Space

When given a learning problem it is important to try and get intuition into "what the data looks like". In the case of a training sample for binary classification, we can plot the different axes and color by the label (Figure 3.2). This task is more difficult for data of higher dimensions, but attempting to imagine it in such cases will help understand what models might fit better to the specific task.

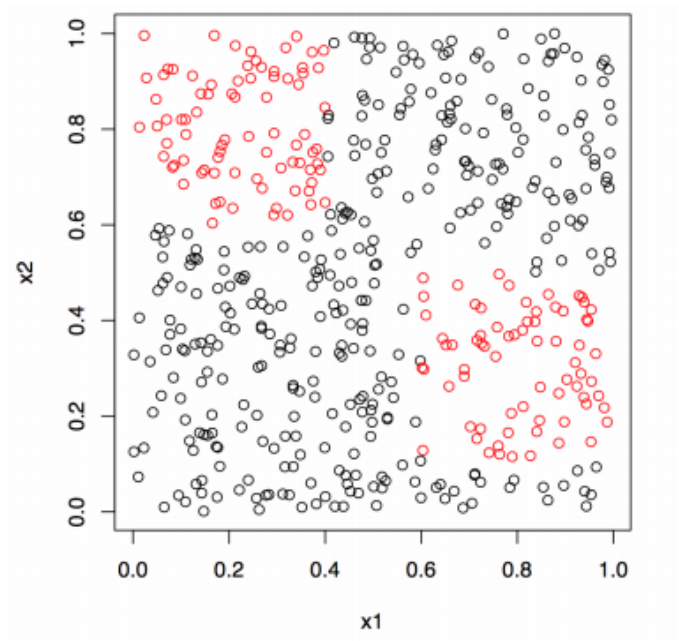


Figure 3.2: Classification training sample in $\mathbb{R}^2$***:*** *Where samples are positioned in space according to the values of their features and color coded by their label.*

3.1.1 Loss Functions

When we discussed regression problems we decided to measure the performance of a given hypothesis using the square loss (and mentioned that we could also use the absolute loss). For classification problems let us consider other loss functions. A very straight forward way to evaluate the performance of a classification predictor is to simply count the number of correctly classified samples. That is, given a prediction rule $h : \mathcal{X} \to \{\pm 1\}$ and a labeled sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, the **misclassification** loss of h on this sample is:

$$L_S(h) \coloneqq \sum_{i=1}^{m} \mathbb{1}[y_i \neq h(\mathbf{x}_i)] = |\{i | y_i \neq h(\mathbf{x}_i)\}| \tag{3.2}$$

3.1.2 Type-I and Type-II Errors

Can there be any problems or issues with the misclassification loss? After all, it just counts the number of times h was wrong - the number of times h misclassified a sample. In practice, there are two kinds of errors the classifier can make, and making each kind of error might have very different implications, or costs. Therefore, simply counting the total number of errors may not be a useful performance measure.

■ **Example 3.2 — Credit Decisions.** Suppose we are building a classifier that predicts whether a bank customer seeking a loan is credit-worthy and will return a given loan or not. We choose the labels such that -1 means "not credit worth - deny loan", and 1 means "credit worthy - approve loan". Denote y_i the true label and $\widehat{y}_i$ the classifier-predicted label of sample i. The two errors this classifier might make have very different consequences:

- If $y_i = -1$ and $\widehat{y}_i = 1$, the classifier predicted that a non-credit-worthy customer will return the loan. If we act on this prediction, and the customer defaults on the loan, the bank loses all the loan sum.
- On the other hand, if $y_i = 1$ and $\widehat{y}_i = -1$, the classifier predicted that a credit-worthy customer, which would have paid the interest and returned the loan in full, is not credit-worthy and should be denied the loan. If we act on this recommendation, the bank loses the interest it would have earned on the loan.

Which of the two errors is more serious? Which of the two errors cost more for the bank? If we could choose which error should we avoid "at all costs" and which error could we "allow to happen", what would we choose? ■

■ **Example 3.3 — Drug safety.** Let us look at a more extreme example to help illustrate this point. We are creating a classifier to predicts whether a certain drug **is safe** to use for a particular person, or **unsafe**/ deadly/ dangerous to use. We choose the labels such that -1 means "unsafe drug - do not use" and 1 means "safe drug - ok to use". Similar to before our errors are:

- If $y_i = -1$ and $\widehat{y}_i$, the classifier recommends to give a drug which is actually potentially deadly.
- If $y_i = 1$ and $\widehat{y}_i$, the classifier recommends that the patient should avoid a drug which is actually safe to use.

■

Therefore, we see that depending on the context of the classification problem, the two kinds of errors can have very different costs. We name the first error, the one we would like to avoid at all costs, the **Type-I error** and the second error as **Type-II error**. By choosing what label is "negative"

and what label is "positive" we essentially defined what error is the Type-I error. As such, given a classification problem we try to choose the "negative" and "positive" labels such that the error we are more concern of (and therefore would like to avoid more) is the Type-I Error. That is, the error of misclassifying a negative sample by predicting it as a positive sample.

Returning to the drug safety example 3.3, we can assign the following meaning to the labels: $y = -1$ (negative) means the new drug is safe to use and $y = 1$ (positive) means the new drug is dangerous. In this case the Type-I error means that we decided not to offer a safe drug. If however we reverse the meaning such that $y = -1$ (negative) means the drug is dangerous and $y = 1$ (positive) means the new drug is safe, then the Type-I error means that we have decided to offer a dangerous drug. In this case this assignment of labels is the more serious of the two kinds of errors we can make.

For the classification problem of "is this email spam or not" how would you choose the labels? What are the two errors a spam detector can make? which one is the one we really want to avoid? So, which of the labels "spam email" and "not spam, valid email" would you label "negative" and which is "positive"?

3.1.3 Measurements of performance

With the decision on "positive" and "negative" labels, we define four basic terms: True Positive (TP), False Positive (FP), True Negative (TN) and False Negative(FN). These terms refer to the prediction made for a sample with respect to its true label. So suppose a sample's true label is $y = -1$ (negative). If a classifier predicts:

- $\hat{y} = -1$ we term this as true negative.
- $\hat{y} = 1$ we term this as false positive.

Now, suppose a sample's true label is $y = 1$ (positive) then if a classifier predicts:

- $\hat{y} = -1$ we term this as false negative.
- $\hat{y} = 1$ we term as true positive.

Therefore, the false negative and false positive cases are the misclassification errors. False positive is what we referred to as Type-I error and false negative is what we called Type-II error. These four options are shown in **??**.

Using these four basic groups we can devise more domain-specific measurements. Denote by P the number of positive samples and N the number of negative samples then:

- The *Error Rate* is the number of misclassification out of all predictions: $(FP+FN)/(P+N)$.
- The *Accuracy* is the number of correct classification out of all predictions: $(TP+TN)/(P+N) = 1-ErrorRate$.
- The *Recall/ Sensitivity/ True-Positive-Rate (TPR)* is the number of truthfully positive predictions out of all positive samples: TP/P.
- The *False-Positive-Rate (FPR)* is the number of of falsely positive predictions out of all negative samples: FP/N.

There are many more measurements that can be defined from the four basic ones presented with different fields using different measurements. In Computer Science we often encounter the TPR and FPR for reasons described below.

3.1.4 Decision Boundaries

Let h be a binary classification rule in $\mathbb{R}^d$. (Suppose, for example, that we used a training sample to select h from some hypothesis class $\mathcal{H}$). We can feed any point $\mathbf{x} \in \mathbb{R}^d$ into h and get one of two classes. This means that we can view $\mathbb{R}^d$ as disjoint union of two sets:

$$\mathbb{R}^d = \{\mathbf{x}|h(\mathbf{x}) = 1\} \biguplus \{\mathbf{x}|h(\mathbf{x}) = 0\}$$

These sets can be very simple (two half-spaces) or very complicated. The boundary between these two sets is called the **decision boundary**: a test sample on one side of the boundary will be classified to one class by h, and a test sample on the other side of the boundary will be classified to the other class.

Different classifies, derived from different hypothesis classes, will generate different decision boundaries (Figure 3.3). Observing these over different data scenarios is helpful to understand is modeled by the different classifiers. It can also help get a qualitative assessment of the bias and variance of the classifiers.

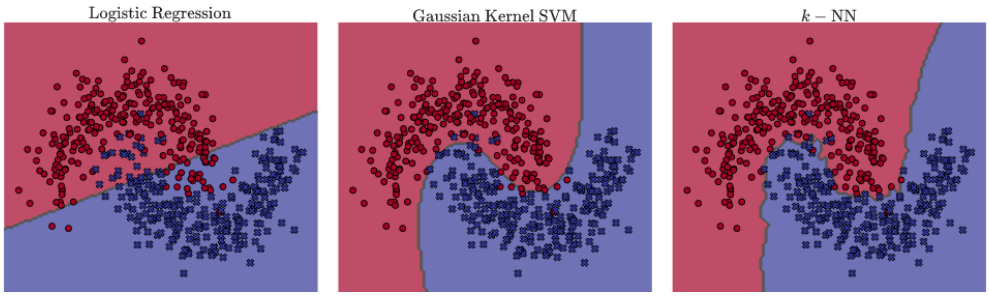


Figure 3.3: Decision Boundaries *of classifiers fitted over moons dataset. Chapter 3 Examples - Source Code*

3.1.5 Studying A New Classifier

For the rest of this chapter we will discuss different sorts of classifiers. As there are numerous types of classifiers, each for tailored for specific data scenarios, it is important to understand how to read about a new classifier. Therefore, when going over the classifiers below keep in mind the following guiding questions:

- How does it model the classification problem? and what are the assumptions made on the data?
- What is the hypothesis class defined? and how does the decision boundary looks like?
- What is the learning principle we use? and how does the algorithm match the learning principle?
- How can the learning principle can be implemented computationally? What is the time complexity of the algorithm and are there any considerations of numeric stability?
- What is done in the training step? and how, given a trained model, to predict for new samples?
- Is the model interpretable? Are we provided with estimations of class probabilities?
- Are we facing a single model or rather a family of models with some parameters for choosing specific models from this family? How do these parameters affect the bias-variance tradeoff?

- When will we decide to use this learning algorithm? What are its advantages and disadvantages?

3.2 Half-Space Classifier

Similar to linear regression, one of the simplest families of classifiers is that of linear classifiers. In these, we are interested in separating a given dataset into two classes using a linear separator function, as seen in Figure 3.4. It will be convenient to work with the class labels of $\mathcal{Y} := \{\pm 1\}$.

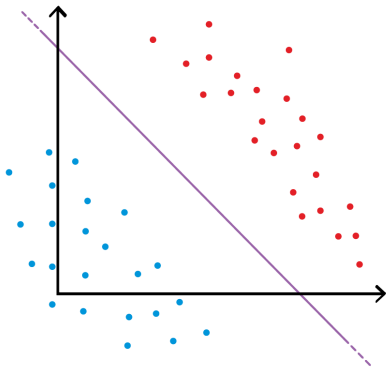


Figure 3.4: Half-space Classification Illustration: *For a domain-set $\mathcal{X} \in \mathbb{R}^2$ the two classes, coded as red and blue colors, are linearly separable .*

Similar to the definition used in linear regression (2.2), the family of linear functions can be described as the set of functions of the form $\mathbf{x} \mapsto \mathbf{x}^\top \mathbf{w} + b,\ \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$. The linearity refers to the functions being linear in the parameters $\mathbf{w}$. Unlike in the regression setup, here we are interested in a mapping to a discrete response value.

Definition 3.2.1 Let $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$. The hyperplane defined by $(\mathbf{w}, b)$ is the set

$$\left\{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle = b, \mathbf{x} \in \mathbb{R}^d\right\}$$

Definition 3.2.2 Let $(\mathbf{w}, b)$ be an hyperplane, so the half-space of $(\mathbf{w}, b)$ is defined as the set

$$\left\{\mathbf{x} | \langle \mathbf{w}, \mathbf{x} \rangle \geq b, \mathbf{x} \in \mathbb{R}^d\right\}$$

or equivalently as $\left\{\mathbf{x} | sign\left(\langle \mathbf{w}, \mathbf{x} \rangle - b\right) \geq 0, \mathbf{x} \in \mathbb{R}^d\right\}$.

Let us define the hypothesis class of half-spaces in $\mathbb{R}^d$. These functions can be thought of as the composition of the *sign* function over the linear functions:

$$\mathcal{H}_{half} := \left\{h_{\mathbf{w},b}(\mathbf{x}) = sign\left(\mathbf{x}^\top \mathbf{w} + b\right) \mid \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\right\} \tag{3.3}$$

So why are functions in the form seen in (3.3) are half-space classifiers? Let us assume at first that

$b = 0$. We can express the domain set as a disjoint union of the following:

$$\mathbb{R}^d = \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} > 0\right\} \biguplus \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} = 0\right\} \biguplus \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} < 0\right\} \tag{3.4}$$

These sets correspond to the open half spaces on either side of the hyperplane $\mathbf{w}^\perp = \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}^\top \mathbf{w} = 0\right\}$ and points on the hyper-plane itself. As such, each vector $\mathbf{w} \in \mathbb{R}^d$ defines a hyper-plane $\mathbf{w}^\perp$ that divides $\mathbb{R}^d$ into two half-spaces.

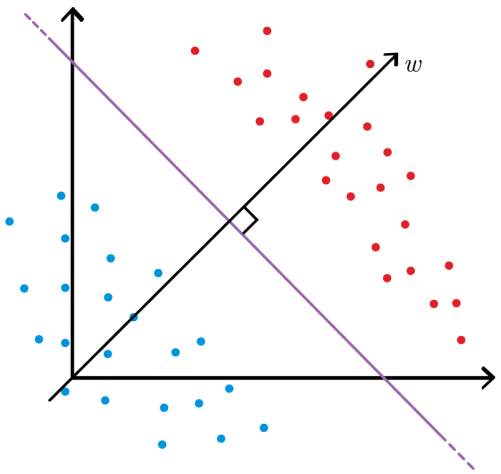


Figure 3.5: *Corresponding Hyperplane to* $\mathbf{w}^\perp$

The case where $b = 0$ is called the **homogeneous** case, as the hyperplane $\mathbf{w}^\perp$ is a linear subspace going through the origin. When $b \neq 0$ the hyperplane does not go through the origin and is called the non-homogeneous case. Recall that we have seen how we could transition from the non-homogeneous to the homogeneous case in the linear regression chapter.

Given a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, we would like to find an hypothesis $h_{\mathbf{w},b} \in \mathcal{H}_{half}$ such that all data points in S that are labeled 1 are on the one side of the hyper-plane and all those labeled -1 are on the other side. To find such an hypothesis we must first make the assumption that the dataset is **linearly separable**. That is, there exists a hyper-plane such that samples of opposing labels are on opposite sides. Mathematically, we assume that

$$\exists \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \quad s.t \quad \forall i \in [m] \quad y_i \cdot sign\left(\langle \mathbf{x}, \mathbf{w} \rangle + b\right) = 1$$

or equivalently since the inner product will be negative for all samples with $y_i < 0$ and positive for all samples with $y_i > 0$:

$$\exists \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \quad s.t \quad \forall i \in [m] \quad y_i \cdot \left(\langle \mathbf{x}, \mathbf{w} \rangle + b\right) > 0 \tag{3.5}$$

Note, that assuming that a given training set is linearly separable is a **realizability assumption**. Namely, the labels are generated by a function in our hypothesis class $\mathcal{H}_{half}$. For simplicity, let us consider the homogeneous case where $b = 0$, since in the non-homoneneous case we can always shift the data by some fixed vector such that the separating hyperplane goes through the origin. So the hypothesis class of linear separators is of the form:

$$\mathcal{H}_{half} := \left\{h_{\mathbf{w}}(\mathbf{x}) = sign\left(\mathbf{x}^\top \mathbf{w}\right) \mid \mathbf{w} \in \mathbb{R}^d\right\} \tag{3.6}$$

3.2.1 Learning Linearly Separable Data Via ERM

To train a model over the defined hypothesis class of homogenous half-spaces ($\mathbf{w} \in \mathbb{R}^d, b = 0$) observe the following: for any hypothesis $h_{\mathbf{w}} \in \mathcal{H}_{half}$, the misclassified training samples are exactly those where $y_i \cdot sign\left(\mathbf{x}^\top \mathbf{w}\right) = -1$ or equivalently $y_i \cdot \mathbf{x}^\top \mathbf{w} < 0$. So defining the loss of a given hypothesis over S is:

$$L_S\left(h_{\mathbf{w}}\right) \coloneqq \sum_{i=1}^{m} \mathbb{1}\left[y_i \cdot \mathbf{x}^\top \mathbf{w} < 0\right] \tag{3.7}$$

Since we are assuming realizability (i.e. that S is linearly separable), we would like to find $h_{\mathbf{w}} \in \mathcal{H}_{half}$ that perfectly separates the training set. Such an hypothesis will be one that achieves $L_S\left(h_{\mathbf{w}}\right) = 0$. In other words, we are applying the ERM principle and seeking for any separating hyperplane $\mathbf{w}^\perp$, corresponding to an hypothesis $h_{\mathbf{w}}$ that minimizes the empirical risk $L_S\left(h_{\mathbf{w}}\right)$.

So the next task is finding a compulationally efficient algorithm to find the desired hypothesis. As we are applying the ERM principle we would like to efficiently compute

$$\hat{\mathbf{w}} \coloneqq \underset{\mathbf{w} \in \mathbb{R}^d}{argmin} L_S\left(h_{\mathbf{w}}\right) \tag{3.8}$$

where since assuming realizability, we know there exists a vector $\mathbf{w}^* \in \mathbb{R}^d$ such that $y_i \left\langle \mathbf{x}_i, \mathbf{w}^* \right\rangle > 0 \quad i = 1, \ldots, m$ (3.5). Notice, that if we define $\bar{\mathbf{w}} = \frac{\mathbf{w}^*}{\gamma}$ for $\gamma = \min_i \left(y_i \left\langle \mathbf{x}, \mathbf{w}^* \right\rangle\right)$ we have that

$$y_i \left\langle \mathbf{x}_i, \bar{\mathbf{w}} \right\rangle = \frac{1}{\gamma} y_i \left\langle \mathbf{x}_i, \mathbf{w}^* \right\rangle \geq 1 \quad i = 1, \ldots, m \tag{3.9}$$

Thus, it is enough to search for a vector $\hat{\mathbf{w}}$ that satisfies

$$y_i \cdot sign\left(\mathbf{x}^\top \hat{\mathbf{w}}\right) \geq 1 \quad \forall i = 1, \ldots, m \tag{3.10}$$

Convex Optimization

In (3.10) we therefore understand that the problem of finding a linear separator is in fact a a problem of finding a vector that satisfies m linear constraints. As these constraints are all convex constraints this is an exaple of what is known as a *convex optimization problem*. Specifically it is a case of a linear program.

Definition 3.2.3 — Optimization Problem. An optimization problem over $\mathbb{R}^d$ has the general form:

$$\begin{array}{ll} minimize & f_0\left(\mathbf{x}\right) \\ subject\,to & f_i\left(\mathbf{x}\right) \leq b_i \quad i = 1, \ldots, n \end{array}$$

where $\mathbf{x}$ is the optimization variable, $f_0 : \mathbb{R}^d \to \mathbb{R}$ is the objective function and $f_i : \mathbb{R}^d \to \mathbb{R}$ are the constraint functions. It is implicitly implied that the optimization problem happens over $dom\left(f_0\right) \subset \mathbb{R}^d$, the domain of f_0.

Then, naturally, a convex optimization problem is an optimization problem as above in which $f_0, f_1, \ldots, f_n$ are all convex functions. When these functions are all linear, this is a linear programming problem

Definition 3.2.4 — Linear Program. An optimization problem is called a Linear Program (LP) if it can be written in the following form:

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{min} & c^\top \mathbf{x} \\ such that & A\mathbf{x} \leq \mathbf{b} \end{array}$$

where $A \in \mathbb{R}^{m\times n}, \mathbf{c} \in \mathbb{R}^n, \mathbf{b} \in \mathbb{R}^m$ are fixed vectors and matrices.

In general, optimization problems are hard to solve computationally. We take special interest in **convex optimization** problems since they have a unique solution, and that solution can be found in computationally tractable ways. A great deal is known about **convex optimization algorithms**, which are iterative numerical algorithms that converge to the solution of a convex optimization problem. There are general solvers, which will solve a convex problem in the general form above, and there are specialized solvers for specific types, or families, of convex optimization problems. A specialized solver is typically preferred, as it leverages some particular structure of the problem to solve it more efficiently, using less space, etc.

Why is convex optimization interesting for machine learning? In supervised learning, we would like to choose a hypothesis $h \in \mathcal{H}$ from our selected hypothesis class, based on some learning principle (such as ERM). Many learning principles are formulated as optimization problems, namely, the h chosen by the learning algorithm is given as the minimizer of some quantity. So implementation of the learning algorithm needs to solve an optimization problem.

Sometimes, our hypothesis class is equivalent to a Euclidean space. When this happens, our learning principle reduces to solving an optimization problem, namely, the hypothesis we choose $h \in \mathcal{H}$ is found as a minimum over $\mathbb{R}^d$ or a subset of $\mathbb{R}^d$ of some objective function, usually a loss function. When this objective is convex, we can use convex optimization algorithms to implement our learning algorithm efficiently.

3.2.2 Solving ERM for Half-Spaces

Returning to the problem of the half-spaces classifier, we have seen that a hyperplane $\mathbf{w}^\perp$ minimizing the empirical risk is in fact a solution to the following linear program:

$$\begin{array}{ll} minimize & 0 \\ subject to & y_i \cdot \langle \mathbf{x}, \mathbf{w} \rangle \geq 1 \quad i = 1, \ldots, m \end{array} \tag{3.11}$$

Such an optimization problem, where a trivial objective, it is a **feasibility** problem. That is, we are looking for any vector which satisfies the constraints. To solve this we can apply some generic solver for linear programs.

3.2.2.1 The Perceptron Algorithm

Another way for finding a separating hyperplane using the ERM principle is by using the Perceptron algorithm, suggested by Frank Rosenblatt in 1958. This is an iterative algorithm that constructs a series of vectors $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \ldots$, where each vector is derived from the vector preceeding it. At each iteration t we search for a sample i which is misclassified by $w^{(t)}$. Then, we update $\mathbf{w}^{(t)}$ by moving it in the direction of thhe misclassified sample $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$.

Algorithm 1 Batch-Perceptron

procedure PERCEPTRON($S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$)
  $\mathbf{w}^{(1)} \leftarrow 0$ ▷ Initialize parameters
  for $t = 1, 2, \ldots$ **do**
    if $\exists i \; s.t. \; y_i \left\langle \mathbf{w}^{(t)}, \mathbf{x}_i \right\rangle \leq 0$ **then** ▷ If there exists a misclassified sample
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$
    else
      return $\mathbf{w}^{(t)}$
    end if
  end for
end procedure

Our goal when using the Perceptron algorithm is to find a vector $\mathbf{w}$ such that $y_i \cdot \mathbf{x}_i^\top \mathbf{w} > 0 \quad i = 1, \ldots, m$. Notice that as

$$y_i \langle \mathbf{w}^{(t+1)}, \mathbf{x}_i \rangle = y_i \langle \mathbf{w}^{(t)} + y_i \mathbf{x}_i, \mathbf{x}_i \rangle = y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle + ||\mathbf{x}_i||^2$$

the update rule of the Perceptron iteratively adjusts the hyperplane to be "more correct" on the i'th sample. It can be shown that in the realizable case the algorithm is guaranteed to terminate, returning a solution that correctly classifies all the samples.

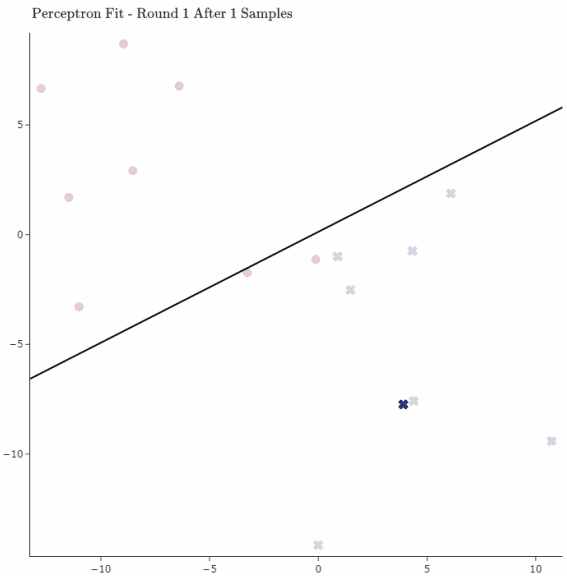


Figure 3.6: Perceptron Fitting: *Fit a separating hyperplane using Perceptron algorithm. Chapter 3 Examples - Source Code*

R The Perceptron algorithm is in fact a simple case of the more general algorithm of Subgradient Descent covered in **??**. Furthermore, we can modify the algorithm is such a way that rather than requiring an entire dataset S, it will each time get a single sample and update based on that one sample. We will encounter this variation in Online Learning **??**.

3.2.3 Learner ID Card

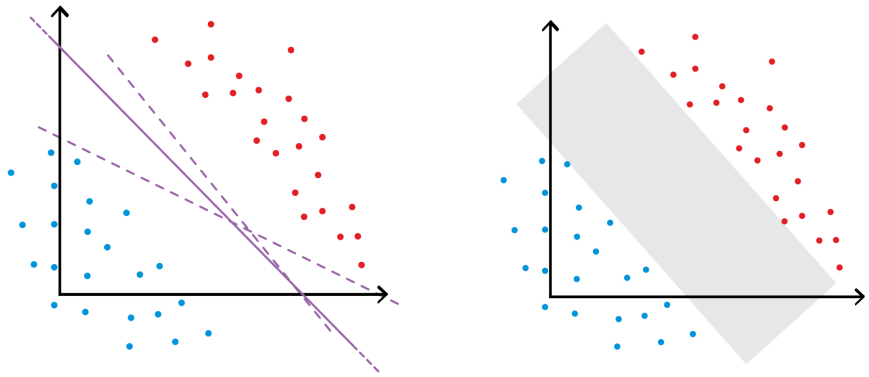
- **Hypothesis class:** the class of linear separators (3.6)
- **Learning principle used for training:** ERM for misclassification loss
- **Computational implementation:** Linear programming or Perceptron
- **Interpretability:** We do not have any specific insight into why a solution was chosen besides it simply satisfying the conditions.
- **Family of models:** No.
- **Storing fitted model:** Fitted model is the vector $\mathbf{w}$ perpendicular to the hyperplane defining the half-space. To store the model we simply store the d coefficients of the vector. In the case of non-homogeneous halfspace we also store the intercept coordinate
- **Prediction of new sample:** $\hat{y}_{new} := sign\left(\mathbf{x}_{new}^{\top}\mathbf{w}+b\right)$
- **When to use:** Since realizability assumption rarely holds this classifier is only used as a simple baseline

3.3 Support Vector Machines (SVM)

When using the half-space classifier seen above, we encounter two problems:

- **Solution Uniqueness:** When we are searching for a separating half-space the solution is not unique. That is, there could be more than a single vector satisfying the constraints of (3.11) and achieving the minimal empirical loss (of zero when assuming realizability or any other positive number if not). As such we are faced with the problem of which one to choose. Figure 3.7 illustrates the existance of multiple separating hyper-planes.

- **Realizability:** A more severe problem rises when we chose to work with the ERM learning principle for selecting the hypothesis, but the data is not linearly separable (non-realizible case). In this case the optimization problem described in 3.11 is computationally hard.



(a) *Algorithm selected hyper-plane (solid line) and additional optional separating hyperplanes (dashed lines)*

(b) *All hyper-planes linearly separating the data*

Figure 3.7: *Illustration of the existance of multiple separating hyper-planes*

Returning to the hypothesis class of non-homogeneous separating half-spaces $\mathcal{H}_{half}$ (3.3), we would like to describe a different learning principle that will be able to cope with both problems above:

finds a unique hyperplane and that can be implemented computationally efficiently even when data is not linearly separable (i.e with polynomial running time given the input).

3.3.1 Maximum Margin Learning Principle

Definition 3.3.1 Let $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ be a hyperplane and $u \in \mathbb{R}^d$. Define the distance between $(\mathbf{w}, b)$ and u by:

$$d\left((\mathbf{w}, b), u\right) := \min_{v:\langle v,w\rangle+b=0} ||u - v||$$

(namely, the Euclidean distance between u and the closest point on the hyperplane)

Definition 3.3.2 — Margin. Let $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ be a hyperplane and $S = u_1, \ldots, u_m \in \mathbb{R}^d$ a set of points. The **margin** of $(\mathbf{w}, b)$ and S is the smallest distance between the hyperplane and any point:

$$M\left((\mathbf{w}, b), S\right) := \min_{i\in[m]} d\left((\mathbf{w}, b), u_i\right)$$

The margin of a given hyperplane with respect to S is therefore the minimal distance between the hyperplane and a sample in S. It seems logical that a hyperplane with a larger margin is more likely to still satisfy all separability constraints even if S is slightly different.

So, the new learning principle is: choose $h_{\mathbf{w},b} \in \mathcal{H}_{half}$ that has the **largest margin** with respect to our training data S. Figure 3.8 illustrates the margins of two different potential hyperplanes. Based on these, we would prefer selecting the hyper-plane that is in the center of the grey area. The vectors closest to the hyperplane determine the margin. They are called **support vectors** and hence this learner's name.

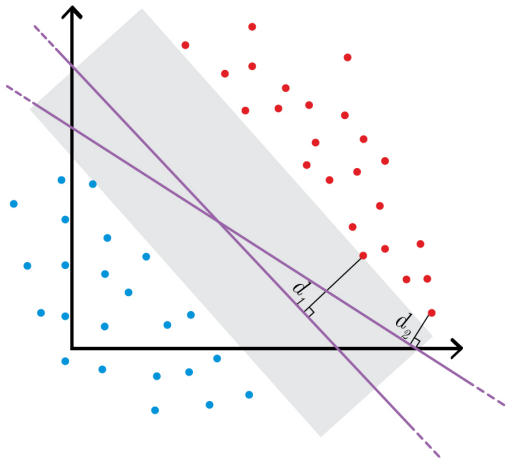


Figure 3.8: *Margin of specified hyper-planes*

3.3.2 Hard-SVM

Let us start with the **realizable case**. To implement our learning principle of maximal margin, we need to search, among all the separating hyperplanes of S, for the hyperplane with maximum margin. Namely, the hypothesis $h_{\mathbf{w},b} \in \mathcal{H}_{half}$ our learner will choose is the solution to the following optimization problem:

$$\begin{array}{ll} \text{maximize} & M\left((\mathbf{w},b),S\right) \\ \text{subject to} & y_i \cdot \left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) > 0 \quad i = 1,\ldots,m \end{array} \tag{3.12}$$

The optimization variables are $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$. Comparing with the linear program of half-spaces (3.11) we see that the constraints are kept, which ensure the hyperplane chosen separates the training sample, but instead of a trivial objective, we seek to minimize the margin.

3.3.2.1 Solving Hard-SVM

So is the Hard-SVM a convex optimization problem? Recall, that by our optimization problem (3.12), we are searching of a separating hyperplane that maximizes the margin from all points. As for any $c > 0$ it holds that $(\mathbf{w},b) = (c\mathbf{w},cb)$ we can w.l.o.g constraint ourselfs to $||\mathbf{w}|| = 1$. This way, each hyperlane has a unique vector $\mathbf{w}$ that corresponds to it.

Definition 3.3.3 Let $\mathbf{x} \in \mathbb{R}^d$ and $B \subseteq \mathbb{R}^d$. The distance from $\mathbf{x}$ to B is:

$$\inf_{\mathbf{v}\in B} ||\mathbf{x}-\mathbf{v}||$$

Lemma 3.3.1 Let $(\mathbf{w},b) \in \mathbb{R}^d \times \mathbb{R}$ be a hyperplane where $||w|| = 1$ and $\mathbf{x} \in \mathbb{R}^d$ then the distance between $\mathbf{x}$ and the hyperplane $(\mathbf{w},b)$ is $|\langle \mathbf{x},\mathbf{w}\rangle + b|$.

Proof. Let $\mathbf{v} := \mathbf{x} - (\langle \mathbf{x},\mathbf{w}\rangle + b)\cdot \mathbf{w}$ be some point in the hyperplane:

$$\begin{array}{rcl} \langle \mathbf{w},\mathbf{v}\rangle + b & = & \langle \mathbf{w}, \mathbf{x} - (\langle \mathbf{x},\mathbf{w}\rangle + b)\cdot \mathbf{w}\rangle + b \\ & = & \langle \mathbf{w},\mathbf{x}\rangle - (\langle \mathbf{x},\mathbf{w}\rangle + b)\,||\mathbf{w}||^2 + b \\ & = & \langle \mathbf{w},\mathbf{x}\rangle - \langle \mathbf{x},\mathbf{w}\rangle - b + b = 0 \end{array}$$

where the distance between $\mathbf{v}$ and $\mathbf{x}$ is:

$$||\mathbf{x}-\mathbf{v}|| = |\langle \mathbf{x},\mathbf{w}\rangle + b| \cdot ||\mathbf{w}|| = |\langle \mathbf{x},\mathbf{w}\rangle + b|$$

To show that this is indeed the distance between $\mathbf{x}$ and the hyperplane, we should show that for any other point $\mathbf{u}$ in the hyperplane the distance between $\mathbf{x}$ and $\mathbf{u}$ is at least as large as between $\mathbf{x}$ and $\mathbf{v}$:

$$\begin{array}{rcl} ||\mathbf{x}-\mathbf{u}||^2 & = & ||\mathbf{x}-\mathbf{v}+\mathbf{v}-\mathbf{u}||^2 \\ & = & ||\mathbf{x}-\mathbf{v}||^2 + ||\mathbf{v}-\mathbf{u}||^2 + 2\langle \mathbf{x}-\mathbf{v},\mathbf{v}-\mathbf{u}\rangle \\ & \geq & ||\mathbf{x}-\mathbf{v}||^2 + 2\langle \mathbf{x}-\mathbf{v},\mathbf{v}-\mathbf{u}\rangle \\ & = & ||\mathbf{x}-\mathbf{v}||^2 + 2\langle (\langle \mathbf{w},\mathbf{x}\rangle + b)\,\mathbf{w},\mathbf{v}-\mathbf{u}\rangle \\ & = & ||\mathbf{x}-\mathbf{v}||^2 + 2(\langle \mathbf{w},\mathbf{x}\rangle + b)\,\langle \mathbf{w},\mathbf{v}-\mathbf{u}\rangle \\ & = & ||\mathbf{x}-\mathbf{v}||^2 + 2(\langle \mathbf{w},\mathbf{x}\rangle + b)\,(\langle \mathbf{w},\mathbf{v}\rangle - \langle \mathbf{w},\mathbf{u}\rangle) \\ & = & ||\mathbf{x}-\mathbf{v}||^2 \end{array}$$

■

So, as the margin between a given hyperplane $(\mathbf{w}, b)$ and a set of points S is the minimal distance between the hyperplane and any point in the set, we derive that our optimization problem is in fact of the form:

$$\begin{aligned} &\underset{(\mathbf{w},b):||\mathbf{w}||=1}{\text{argmax}} \quad \min_{i\in[m]} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| \\ &\text{subject to} \quad y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0 \quad i = 1, \ldots, m \end{aligned} \tag{3.13}$$

While the constraints enforce $\mathbf{w}$ to define a separating hyperplane, the objective will make us choose a separating hyperplane with the maximal margin. We further simplify the optimization problem. Consider a *feasible* solution $\mathbf{w}$ to the problem (i.e. that satisfies all constraints). It holds that $|\langle \mathbf{x}_i, \mathbf{w} \rangle + b| = y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b)$. Hence, we can rewrite (3.13) as:

$$\begin{aligned} &\underset{(\mathbf{w},b):||\mathbf{w}||=1}{\text{argmax}} \quad \min_{i\in[m]} y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \\ &\text{subject to} \quad y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0 \quad i = 1, \ldots, m \end{aligned} \tag{3.14}$$

Notice that the constraints are now redundant. If $\mathbf{w}$ is infeasible then $\min_i y_i \left(\mathbf{x}_i^\top \mathbf{w} + b\right) < 0$, achieving a lower objective than any feasible solution. Therefore, we can re-write the problem as:

$$\underset{(\mathbf{w},b):||\mathbf{w}||=1}{\text{argmax}} \quad \min_{i\in[m]} y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \tag{3.15}$$

And lastly, we notice that we can represent (3.15) as a norm minimization problem instead of margin maximization problem:

Claim 3.3.2 Consider the following optimization problem:

$$\underset{(\mathbf{w},b)}{\text{argmin}} \ ||\mathbf{w}||^2 \qquad \text{subject to} \quad y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 \quad i = 1, \ldots, m \tag{3.16}$$

If $(\mathbf{w}^*, b^*)$ is an optimal solution to (3.16) then $\left(\hat{\mathbf{w}}, \hat{b}\right)$ is an optimal solution to (3.15), where $\hat{\mathbf{w}} = \frac{\mathbf{w}^*}{||\mathbf{w}^*||}, \hat{b} = \frac{b^*}{||\mathbf{w}^*||}$.

Proof. Let $(\mathbf{w}, b)$ be a feasible solution to (3.15) and denote the margin achieved by $(\mathbf{w}, b)$ by γ then:

$$y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq \gamma \quad i = 1, \ldots, m$$

Since $\gamma = \min_i y_i (\langle \mathbf{x}_i, \mathbf{w} \rangle + b)$ it holds that:

$$y_i \left(\left\langle \mathbf{x}_i, \frac{\mathbf{w}}{\gamma} \right\rangle + \frac{b}{\gamma} \right) \geq 1 \quad i = 1, \ldots, m$$

meaning that $(\frac{\mathbf{w}}{\gamma}, \frac{b}{\gamma})$ is a feasible solution to (3.16). Let $(\mathbf{w}^*, b^*)$ be an optimal solution for (3.16). As such, it achieves the minimal norm out of all feasible solutions and specifically it means that:

$$||\mathbf{w}^*|| \leq \left|\left| \frac{\mathbf{w}}{\gamma} \right|\right| = \frac{1}{\gamma} ||\mathbf{w}|| = \frac{1}{\gamma}$$

where the last equality is due to $(\mathbf{w}, b)$ being a feasible solution to (3.15) and therefore $\mathbf{w}$ a unit vector. Consider $(\hat{\mathbf{w}}, \hat{b})$ achieved from $(\mathbf{w}^*, b^*)$. It follows that for all $i \in [m]$:

$$y_i \left(\langle \mathbf{x}_i, \hat{\mathbf{w}} \rangle + \hat{b} \right) = y_i \left(\left\langle \mathbf{x}_i, \frac{\mathbf{w}^*}{||\mathbf{w}^*||} \right\rangle + \frac{b^*}{||\mathbf{w}^*||} \right) = \frac{1}{||\mathbf{w}^*||} y_i (\langle \mathbf{x}_i, \mathbf{w}^* \rangle + b^*) \geq \frac{1}{||\mathbf{w}^*||} \geq \gamma$$

with $\hat{\mathbf{w}}$ being a unit vector. Thus, $(\hat{\mathbf{w}}, \hat{b})$ achieves a higher or equal objective to (3.15) from any feasible solution, concluding optimality. ■

This means in fact that maximizing the margin is equivalent to minimizing the size of the hyperplane.

Definition 3.3.4 — Quadratic Program. An optimization problem is called a Quadratic Program (QP) if it can be written in the following form:

$$\begin{array}{ll} \underset{\mathbf{w}\in\mathbb{R}^n}{min} & \frac{1}{2}\mathbf{w}^\top Q\mathbf{w}+\mathbf{a}^\top\mathbf{w} \\ \text{such that} & A\mathbf{w}\leq\mathbf{d} \end{array}$$

where $Q\in\mathbb{R}^{n\times n}, A\in\mathbb{R}^{m\times n}, \mathbf{a}\in\mathbb{R}^n, \mathbf{d}\in\mathbb{R}^m$ are fixed vectors and matrices.

The optimization problem written in (3.16) is a **Quadratic Program (QP)** for which there exist efficient solvers. By using them to solve problem (3.16) we can obtain an optimal solution for the Hard-SVM optimization problem.

R But so how is it that minimizing $||\mathbf{w}||^2$ is equivalent to maximizing the margin? Let us denote the width of the total margin (i.e. the sum of margin from both sides) by l, and let x_+ and x_- be the positive- and negative support vectors . To calculate the value of l we will project the vector $x_+ - x_-$ onto the normalized normal $\mathbf{w}$:

$$\begin{aligned} l &= \left\langle x_+ - x_-, \frac{\mathbf{w}}{||\mathbf{w}||}\right\rangle \\ &= (\langle x_+,\mathbf{w}\rangle - \langle x_-,\mathbf{w}\rangle)\,||\mathbf{w}|| \\ &= (1-b-(-1-b))\,/\,||\mathbf{w}|| \\ &= 2/||\mathbf{w}|| \end{aligned}$$

where support vectors satisfy $y_i(\langle\mathbf{w},\mathbf{x}_i\rangle+b)$ and that for positive samples $y_i=1$ and negative samples $y_i=-1$. This shows how minimizing $||\mathbf{w}||$ maximizes l.

3.3.3 Soft-SVM

The basic assumption of Hard-SVM is that the training sample is *linearly separable*, that is, that the realizability assumption holds. If that is not the case then the optimization problem has no solutions as for any candidate $(\mathbf{w},b)$ at least one of the constraints $y_i\cdot\left(\mathbf{x}_i^\top\mathbf{w}+b\right)\geq 1$ cannot be satisfied.

However, what if the training sample is *almost* linearly separable? That is, what if most of the samples are linearly separable with only a few violating the constraints by "not too much"? Recall that if $y_i\cdot\left(\mathbf{x}_i^\top\mathbf{w}+b\right)<0$ then sample $\mathbf{x}_i$ is on the "wrong side" of the hyperplane. This means that:

$$\exists\,\xi_i>0 \quad s.t. \quad y_i\cdot\left(\mathbf{x}_i^\top\mathbf{w}+b\right)\geq 1-\xi_i$$

Therefore, sample $\mathbf{x}_i$ is on the "wrong" side of the **margin** by an amount proportional to ξ_i (Figure 3.9). To allow training samples to violate the constraints "a little", we modify the optimization problem to:

$$\begin{array}{ll} \text{minimize} & ||\mathbf{w}||^2 \\ \text{subject to} & \begin{cases} y_i\cdot\left(\mathbf{x}_i^\top\mathbf{w}+b\right)\geq 1-\xi_i & i=1,\ldots,m \\ \xi_i\geq 0 \quad\wedge\quad \frac{1}{m}\sum_{i=1}^m\xi_i\leq C \end{cases} \end{array} \tag{3.17}$$

where $C > 0$ is a constant we specify. The variables $\xi_1, \ldots, \xi_m$ are new auxiliary variables we introduce (sometimes known as slack variables). Notice that the larger we choose C to be, the more violations of margin we allow. On the one hand, we want to allow "noisy" samples to violate the margin, so the hyperplane will ignore them. On the other hand, if we allow too many violations, we lose touch with the training sample and its structure. This is exactly the bias-variance trade-off: the larger C, the more freedom the learner has to "chase after the training sample".

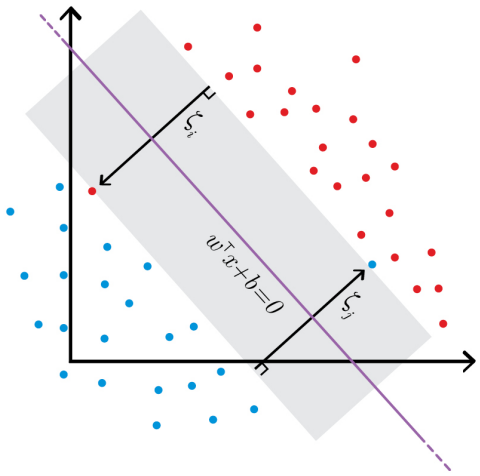


Figure 3.9: *Slack variables of data-points that are on the "wrong" side of the hyper-plane.*

Instead of specifying C directly, we often prefer working with a slightly different optimization problem, where instead of constraining the value of $\frac{1}{m}\sum \xi_i$ we jointly minimize the norm of $\mathbf{w}$ (related to the margin) and the average of ξ_i (corresponding margin violations).

$$\begin{array}{ll} \underset{\mathbf{w},b,\{\xi_i\}}{\text{minimize}} & \lambda \left|\left|\mathbf{w}\right|\right|^2 + \frac{1}{m}\sum_{i=1}^m \xi_i \\ \text{subject to} & y_i \cdot \left(\mathbf{x}_i^\top \mathbf{w} + b\right) \geq 1 - \xi_i,\ \xi_i \geq 0 \quad i = 1, \ldots, m \\ & \lambda \geq 0 \end{array} \tag{3.18}$$

To simplify the above optimization problem let use define the **hinge** loss function:

$$\ell^{hinge}(a) = \max\left\{0, 1-a\right\},\ a \in \mathbb{R} \tag{3.19}$$

Claim 3.3.3 Given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ and hyperplane $(\mathbf{w}, b)$, the Soft-SVM optimization problem (**??**) is equivalent to

$$\underset{\mathbf{w},b}{\text{argmin}} \left(\lambda \left|\left|\mathbf{w}\right|\right|^2 + L_S^{hinge}((\mathbf{w},b))\right)$$

where $L_S^{hinge}((\mathbf{w},b)) := \frac{1}{m}\sum \ell^{hinge}\left(y_i \cdot \mathbf{x}_i^\top \mathbf{w}\right)$

Proof. Given a specific hyperplane $(\mathbf{w},b)$ consider the minimization over $\xi_1,\ldots,\xi_m$. Since we defined the auxiliary variables to be nonnegative, the optimal assignment of ξ_i is

$$\xi_i := \begin{cases} 0 & y_i\left(\mathbf{x}_i^\top \mathbf{w}+b\right) \\ 1-y_i\left(\langle \mathbf{x}_i,\mathbf{w}\rangle+b\right) > 1 & otherwise \end{cases}$$

Thus $\xi_i = \ell^{hinge}\left(y_i\left(\mathbf{x}_i^\top \mathbf{w}+b\right)\right)$ ■

The hyper-parameter λ controls the trade-off between the norm of $\mathbf{w}$ and the violations of margin.

- The larger λ, the less sensitive the solution will be to the term $\frac{1}{m}\sum_{i=1}^m \xi_i$, and will allow more violations.
- The smaller λ, the more sensitive and will allow less violations.

Therefore when we consider the parameter λ in (3.18) (or equivalently C in (3.17)), we are in fact considering different learners within a **family of learners**, where each member of the family is specified by a specific value of λ (or C). These different family members can be placed somewhere along the hypothesis complexity axis. Thus, changing the value of λ (or C) moves us along the bias-variance tradeoff. λ is known as a **regularization parameter**. This topic is covered in section 6.1.

3.3.4 Learner ID Card

- **Hypothesis class:** the class of non-homogeneous linear separators (3.3)
- **Learning principle used for training:** Maximal margin
- **Computational implementation:** Quadratic Program
- **Interpretability:** Retrieved solution does not provide meaningful insights regarding predictions
- **Family of models:** The Soft-SVM learner provides us with a set of models, indexed by the regularization parameter $\lambda \in [0,\infty)$
- **Storing fitted model:** Fitted model is the vector $\mathbf{w}$ perpendicular to the hyperplane defining the half-space as well as the intercept coordinate
- **Prediction of new sample:** $\hat{y}_{new} := sign\left(\mathbf{x}_{new}^\top \mathbf{w}+b\right)$
- **When to use:** By itself this learner should be used as a simple baseline. However, after embedding the data in some high-dimensional space (kernelization) this becomes a powerful learner (chapter 8)

3.4 Logistic Regression

3.4.1 A Probabilistic Model For Noisy Labels

Let us revisit the model of linear regression. Recall that when assuming Gaussian errors ((2.1.4)) we modeled the relation $\mathcal{X} \to \mathcal{Y}$ as $\mathbf{y} = \mathbf{X}\mathbf{w}+\varepsilon$ and $\varepsilon \sim \mathcal{N}\left(0,\sigma^2 I_m\right)$. Notice that as ε is a random variable, $\mathbf{y}$ too is a random variable distributing as a multi-variate Gaussian:

$$\mathbf{y} \sim \mathcal{N}\left(\mathbf{X}\mathbf{w},\sigma^2 I_m\right) \tag{3.20}$$

Focusing on a pair $(\mathbf{x}_i,y_i)$, we can think of the above as the **conditional probability** of y_i given $\mathbf{x}_i$:

$$p\left(y_i|\mathbf{x}_i,\mathbf{w}\right) = \mathcal{N}\left(y_i|\phi_{\mathbf{w}}\left(\mathbf{x}_i\right),\sigma^2\right) \quad \text{where} \quad \phi_{\mathbf{w}}\left(\mathbf{x}\right) = \mathbf{x}^\top \mathbf{w} \tag{3.21}$$

where the notation of $\mathcal{N}(y_i|\mathbf{x}_i,\mathbf{w})$ means the probability of observing the response y_i for the feature vector of $\mathbf{x}_i$ and coefficients vector $\mathbf{w}$. We also condition on $\mathbf{w}$ (though is not a random variable) to explicitly state the dependence on the model parameters. In other words, we assumed that each sample $(\mathbf{x},y)$ is such that the **expected value** of the label y is linear in $\mathbf{x}$. As we are dealing with a regression model and $y_i \in \mathbb{R}$, the support of the random variable $y_i|\mathbf{x}_i,\mathbf{w}$ is $\mathbb{R}$.

Let us adapt the model above to fit for classification problems. We would like to assume that y_i distributes **Bernoulli** with a probability $p(\mathbf{x}_i)$ that somehow relates with $\mathbf{x}_i$, and captures how likely is y_i of being 1:

$$p(y_i|\mathbf{x}_i) = Ber(y_i|p(\mathbf{x}_i)) \tag{3.22}$$

How shall $p(\mathbf{x}_i)$ relate with $\mathbf{x}_i$? Unlike the linear regression model, we cannot assume a linear function $\phi_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^\top\mathbf{w}$ as $\phi_{\mathbf{w}} \in \mathbb{R}$ while $p(\mathbf{x}_i)$ is restricted to $[0,1]$. Instead, we would like to choose some **link** function $\phi_{\mathbf{w}} : \mathbb{R} \to [0,1]$ that is monotone increasing and maps $(-\infty,\infty)$ bijectively to $(0,1)$. Define the relation to be:

$$p(y_i|\mathbf{x}_i,\mathbf{w}) = Ber(y_i|\phi_{\mathbf{w}}(\mathbf{x}_i)), \quad \phi_{\mathbf{w}} := sigm\left(\mathbf{x}^\top\mathbf{w}\right) \tag{3.23}$$

where *sigm* is the **sigmoid** function, also known as the **logit** function:

$$sigm(\mathbf{a}) := \frac{e^{\mathbf{a}}}{e^{\mathbf{a}}+1} \tag{3.24}$$

This function is indeed monotone increasing and maps $(-\infty,\infty)$ bijectively to $(0,1)$:

- As $\mathbf{x}^\top\mathbf{w} \to -\infty$ then $sigm\left(\mathbf{x}^\top\mathbf{w}\right) \to 0$. This means that it is "very unlikely" that the label is 1: $p(y_i = 1|\mathbf{x}_i,\mathbf{w}) \to 0$.
- As $\mathbf{x}^\top\mathbf{w} \to \infty$ then $sigm\left(\mathbf{x}^\top\mathbf{w}\right) \to 1$. This means that it is "very likely" that the label is 1: $p(y_i = 1|\mathbf{x}_i,\mathbf{w}) \to 1$.

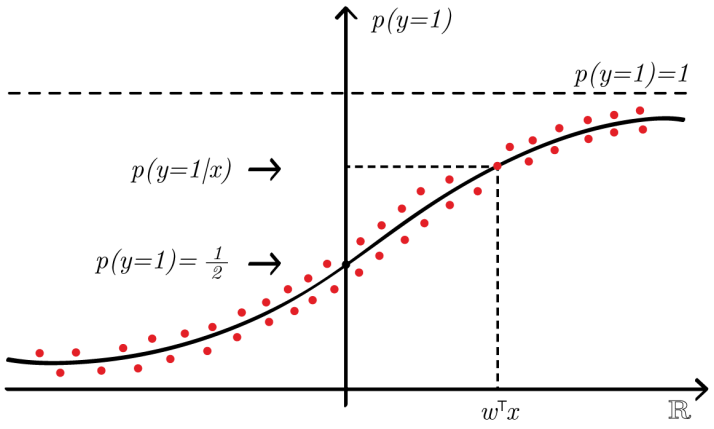


Figure 3.10: Illustration of fitted logit function *for values corresponding to* $\mathbf{x}^\top\mathbf{w}$.

R In (3.23) we modeled the logistic regression model for binary classification problems. Notice that the Bernoulli distribution can be seen as a private case of the Multinomial distribution $Multinomial(p_1,\ldots,p_K)$, $\sum_p i = 1, 0 \le p_i \le 1$. We can expand the above logistic regression model to fit multi-classification problems by extending the sigmoidal function to what is known as the softmax function $\sigma(\mathbf{a}) = e^{\mathbf{a}_i} / \sum_{j=1}^{K} e^{\mathbf{a}_j}$

3.4.1.1 The Hypothesis Class

So we would like to define the hypothesis class of logistic regression as:

$$\mathcal{H}_{logistic} \coloneqq \left\{ h_{\mathbf{w}}(\mathbf{x}) = sigm\left(\mathbf{x}^{\top}\mathbf{w}\right) | \mathbf{w} \in \mathbb{R}^{d+1} \right\} \tag{3.25}$$

where $\mathbf{w} \in \mathbb{R}^{d+1}$ since we incorporate the intercept variable into $\mathbf{w}$ (and a zeroth coordinate of 1 to $\mathbf{x}$) similar to the way we did in the linear regression hypothesis class. Notice that the hypotheses are defined $h_{\mathbf{w}} : \mathbb{R}^{d+1} \to [0,1]$ and not $h_{\mathbf{w}} : \mathbb{R}^{d+1} \to \{0,1\}$ as required for classification problems. Since $\{0,1\} \subset [0,1]$, we can use the training sample to select a function in $\mathcal{H}_{logistic}$. This means we will be able to train a model, but how will we predict over new samples? Suppose our learner chose some $h_{\mathbf{w}} \in \mathcal{H}_{logistic}$. As we think of $h_{\mathbf{w}}(\mathbf{x})$ as an estimate of the probability that the label corresponding to $\mathbf{x}$ is 1, we can use it for classification. If $h_{\mathbf{w}}(\mathbf{x})$ is low the label is likely to be 0. If $h_{\mathbf{w}}(\mathbf{x})$ is high, the label is likely to be 1. Choosing some **cutoff** value $\alpha \in [0,1]$, our class prediction will be: $\widehat{y} \coloneqq \mathbb{1}[h_{\mathbf{w}}(\mathbf{x}) > \alpha]$. To choose a fitting value for α we can calculate the Type-I and Type-II errors (subsection 3.1.2) of the classifier and plot its ROC curve (3.4.4)

3.4.1.2 Learning Via Maximum Likelihood

Once we have defined the logistic regression model (3.23) and hypothesis class (3.25), we would like to come up with a learner. To do so we will use the *maximum likelihood principle*. Recall, that by the maximum likelihood principle, we estimate the parameters (the desired hypothesis) as those that have the highest probability, given the data.

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be our sample of independent observations, assuming that $\mathbf{y}_i \sim Ber(\phi_{\mathbf{w}}(\mathbf{x}))$ where ϕ is the logistic function. Therefore, the likelihood of $\mathbf{w} \in \mathbb{R}^{d+1}$ is:

$$\begin{aligned}
\mathcal{L}(\mathbf{w}|\mathbf{X},\mathbf{y}) &= \mathbb{P}(y_1,\ldots,y_m|\mathbf{X},\mathbf{w}) \\
&= \prod \mathbb{P}(y_i|\mathbf{x}_i,\mathbf{w}) \\
&= \prod_{i:y_i=1} \mathbb{P}(y_i|\mathbf{x}_i,\mathbf{w}) \cdot \prod_{i:y_i=0} \mathbb{P}(y_i|\mathbf{x}_i,\mathbf{w}) \\
&= \prod_{i:y_i=1} p_i(\mathbf{w}) \cdot \prod_{i:y_i=0} (1-p_i(\mathbf{w})) \\
&= \prod p_i(\mathbf{w})^{y_i} (1-p_i(\mathbf{w}))^{1-y_i}
\end{aligned} \tag{3.26}$$

where $p_i(\mathbf{w}) = \phi_{\mathbf{w}}(\mathbf{x}_i)$. Since the log function is monotone increasing we can maximize the log-likelihood $\ell(\mathbf{w}) \coloneqq \log \mathcal{L}(\mathbf{w})$ instead:

$$\begin{aligned}
\ell(\mathbf{w}|\mathbf{X},\mathbf{y}) &= \sum_{i=1}^m \left[y_i \log(p_i(\mathbf{w})) + (1-y_i)\log(1-p_i(\mathbf{w})) \right] \\
&= \sum_{i=1}^m \left[y_i \log\left(\frac{e^{\mathbf{x}_i^{\top}\mathbf{w}}}{1+e^{\mathbf{x}_i^{\top}\mathbf{w}}}\right) + (1-y_i)\log\left(\frac{1}{1+e^{\mathbf{x}_i^{\top}\mathbf{w}}}\right) \right] \\
&= \sum_{i=1}^m \left[y_i \cdot \mathbf{x}_i^{\top}\mathbf{w} - \log\left(1+e^{\mathbf{x}_i^{\top}\mathbf{w}}\right) \right]
\end{aligned} \tag{3.27}$$

And therefore, choosing the function $h \in \mathcal{H}_{logistic}$ by applying the maximum likelihood principle means that:

$$\hat{\mathbf{w}} \coloneqq \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{argmax} \sum_{i=1}^m \left[y_i \cdot \mathbf{w}^{\top}\mathbf{x}_i - \log\left(1+e^{\mathbf{w}^{\top}\mathbf{x}_i}\right) \right] \tag{3.28}$$

R Instead of deriving the learner using the maximum likelihood principle, we could derive it using the ERM principle with the following loss function: $\ell(h_{\mathbf{w}}) := \log(1+\exp(-y\langle \mathbf{w},\mathbf{x}\rangle))$. We would have reached the same optimization expression.

3.4.2 Computational Implementation

Now that we have defined the hypothesis class and an optimization problem to find the desired hypothesis, the next step is finding an efficient algorithm to solve it. By working with the logistic function, the resulting log-likelihood expression is **concave** function of the optimization variable $\mathbf{w}$. This means, that instead of solving the maximization problem (3.28) we can solve the minimization of minus the log-likelihood, which is convex. As such, there are general algorithms for finding the minima of such functions.

While there is no closed form for the maximizer $\hat{\mathbf{w}}$, as logistic regression is a very useful learner, there is a custom iterative algorithm that usually converges quickly to $\hat{\mathbf{w}}$. This algorithm is based on **Newton-Raphson** iterations.

3.4.3 Interpretability

One important properly of the logistic regression learner is interpretability both in the sense of which features were important for the model and why was a certain prediction given. When working with $\mathcal{X} = \mathbb{R}^d$, we gather many feature, and might think that some of them are important for prediction while others less. Similar to linear regression, we are able to ask "which features were important" for the model by simply investigating the entries of the fitted coefficients vector $\mathbf{w}$. Features corresponding to coefficients of values close to zero have a small impact on prediction and therefore these features are less important for the model. Features corresponding to coefficients of values far from zero have a large impact on prediction and are therefore important for the model.

Then, for a given sample $\mathbf{x}$, by looking at entries corresponding to important features we can understand why the model predicted $\hat{y}$. If in entries of $\mathbf{x}$ corresponding important features there are large (positive or negative) values, they will have much influence the outcome. If these values are of same sign as of the coefficients then the expression $\mathbf{x}^\top\mathbf{w}$ will be larger, increasing the likelihood of the prediction being 1. If these values are of opposite signs to the coefficients then the expression $\mathbf{x}^\top\mathbf{w}$ will become smaller, decreasing the likelihood of the prediction being 1.

3.4.4 Predictions Over New Samples & The ROC Curve

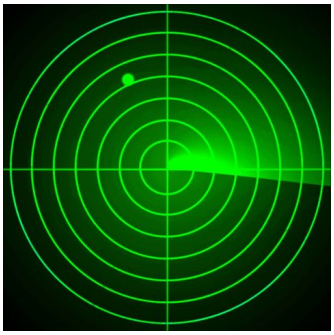
As we will encounter later in this chapter, in many classification scenarios we face the following question: Suppose we trained some classifier $h \in \mathcal{H}$ and derive classifications by the following rule: for some cutoff value $\alpha \in [0,1]$

$$\hat{y} := \begin{cases} 1 & h(\mathbf{x}) > \alpha \\ 0 & h(\mathbf{x}) \leq \alpha \end{cases}$$

How do we choose α? There is an important **tradeoff** in the selection of α. If we set α to be very high we are mainly going to predict 0. By doing so we are **less** likely to have false-positives (which is the error we try to avoid at all cost), for which we are pleased. However, at the same time, we are bf more likely to gave false-negatives. So by setting α too high we might have low a FPR but "miss" (misclassify) most of the positive samples. On the other extreme, if we set α to be very low we are

mainly going to predict 1. So we will be **more** likely to have false-positives, but at the same time will be **less** likely to have false-negatives. So if we set α too low, we might have a high FPR but "catch" (correctly classify) most of the positive samples. Therefore, we see that changing $\alpha \in [0,1]$ governs some trade-off between the chances of making a Type-I error (false-positives) and correctly classifying positive samples.

This trade-off was first studied during World War II, when radar was invented. The designer of the radar had to choose when to put a green dot on the radar, indicating a target detected there. Sometime radar waves would bounce off back from clouds or birds, and the designer had to choose a **threshold** α. If the radar pulse returning is stronger than α, the radar screen would show a green dot. If weaker than α, no dot. Now, if α is set too low (say $\alpha = 0.1$), the screen would be full of a thousand green dots - since any bird or cloud (with, say, $h(\mathbf{x} = 0.2)$) would be classified as **positive**, a target. So that the radar will be full of false positives, false targets, and will be useless. On the other hand, if α is set too high (say $\alpha = 0.9$) then enemy airplanes (with, say, $h(\mathbf{x} = 0.8)$) will not appear on the screen, since they will be classified by mistake as birds, and the radar again would be useless.



The radar engineers developed a way to visualize this tradeoff, which is still used today in machine learning. After training some linear model (choosing some hypothesis $h \in \mathcal{H}$) we make a grid of values of $\alpha \in [0,1]$. For each value of α we create a classifier by thresholding h at α, and calculate the number of Type-I and Type-II errors the classifier makes over a test sample that was not used for training. We plot a parametric curve of TPR (true positive rate) against FPR (false positive rate) when α is the parameter. This curve is called the **Receiver Operating Characteristic (ROC)** curve. It is continuous, increasing and goes from $(0,0)$ in the FPR-TPR plane (for $\alpha = 0$ we classify everything as negative, so no false positives and not true positives) to $(1,1)$ (for $\alpha = 1$ we classify everything as positive, so false positive rate is 1 - we make every possible Type-I error - and also true positive rate is 1 - we "catch" all the positive samples).
Convince yourself that if the ROC curve is a linear line from $(0,0)$ to $(1,1)$, the classifier is just a random guess. If a classifier has an ROC curve that is closed to this linear line, it's a poor classifier. Now convince yourself that if the ROC curve rises sharply from $(0,0)$, for example makes a "jump" to $(FPR = 0.1, TPR = 0.9)$, it's a good classifier - we are able to correctly detect 0.9 of the positive samples at the price of 0.1 false positive rate.

Plotting the ROC curve of a classifier has a few different uses:

- **Tuning α:** It allows us to see the tradeoff, provided by the classifier, between Type-I errors and correct detection of positive samples, so we can choose the tuning of α we would like to work with for the actual prediction.

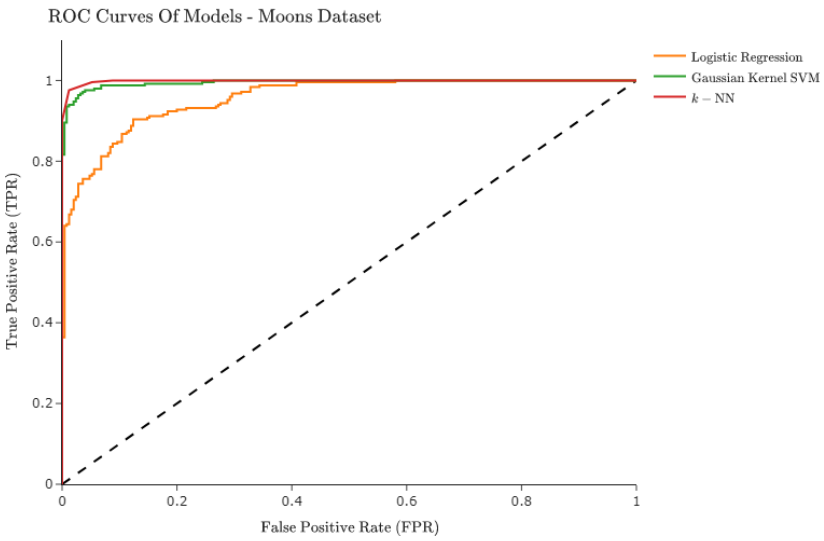


Figure 3.11: ROC Curve *of classifiers fitted over moons dataset. Chapter 3 Examples - Source Code*

- **AUC - Area Under Curve:** A performance measure for the tradeoff itself. This performance measure evaluates the prediction rule h we chose without having to decide on α - it measures the quality of the **tradeoff** provided by h, a tradeoff from which we must choose a specific point in order to actually classify new samples. AUC is simply the **definite integral** of the ROC curve on the segment $[0,1]$ - the area under the AUC curve. As mentioned above, AUC around $1/2$ means that h is poor - more specifically, that the **tradeoff** provided by h is poor. AUC is bounded from above by 1, so an AUC close to 1 means h offers an excellent trade-off, and in this case we expect to be able to find a cutoff α that gives a classifier with very few false-positives and very high detection rate (true positive rate).
- **Comparing candidate rules:** Suppose we have a couple of candidate rules h_1, h_2 (or more). For example, maybe we trained some classifier on the same training sample with different features, or maybe we trained two different types of classifiers over the same data, and we are wondering which one to use. Now we have a problem - we can't turn h_i into an actual classification rule without choosing a cutoff α_i, but would like to compare h_1 to h_2 without committing to a cutoff - to compare the tradeoff offered by h_1 to that offered by h_2. It is very useful here to plot the two ROC curves of h_1 and h_2 on a single axis - and visually compare the tradeoffs they offer.

3.4.5 Learner ID Card

- **Hypothesis class:** The composite of the sigmoidal function over the linear functions (3.25)
- **Learning principle used for training:** Maximum likelihood
- **Computational implementation:** Specialized iterative method based on Netwon-Raphson iterations, or a general convex solver. When using a general convex solver we must pay attention to the effective rank of the regression matrix, similar to linear regression. Near-singular regression matrices will lead to numerical instabilities
- **Interpretability:** Given a fitted model we can interprate which features drive the classification as well as understand why a given sample was predicted as it was

- **Family of models:** As seen in section 6.1 it is possible to add regularization terms to control the bias-variance properties of the fitted model
- **Storing fitted model:** Store the regression coefficients vector $\mathbf{w}$
- **Prediction of new sample:** To perform predictions we must specify a thresholding parameter $\alpha \in (0,1)$. Once we chose a value of α then prediction is performed by $\widehat{y}_{new} := \mathbb{1}\left[sigm\left(\mathbf{x}_{new}^{\top}\mathbf{w}+b\right) \geq \alpha\right]$
- **When to use:** The logistic regression learner, especially when adding regularization terms, is a powerful learner. It is always good to try it, especially when classes are more or less balanced.

For the logistic regression learner we have adapted the hypothesis class of linear regression by composing it with the sigmoid function:

$$\mathcal{H}_{logistic} = \left\{\mathbf{x} \mapsto sigm\left(\mathbf{x}^{\top}\mathbf{w}\right)\right\}$$

Then, we have derived an optimization problem using the maximum likelihood principle (3.28). To computationally implement the learner there are specialized iterative methods based on Newton-Raphson iterations, or general convex solvers. It is important to note that just like linear regression we must pay attention to the effective rank of the regression matrix. Near-singular matrices will cause numerical problems.

Given trained model, we have to specify a threshold parameter α, which will provide some good tradeoff between the FPR and TPR. Once we have specified α prediction of a new sample is given by $\hat{y} := \mathbb{1}\left[sigm\left(\mathbf{x}_{new}^{\top}\mathbf{w}\right) \geq \alpha\right]$.

3.5 Nearest Neighbors

Nearest Neighbors classifiers are a popular, simple and effective learner in which we predict a sample's response based on a set of "nearest" training samples. This classifier is **not** based on the paradigm of a hypothesis class and learning principle. It is a *model free* learner and has no training stage. Instead, when given a training set, we store it in some manner. Then, when we are given a new sample to predict its response we "simply" find the subset of training samples nearest to the new sample, with respect to some measure of distance, and make a prediction based on the responses of those neighbor samples.

R This family of leaners are part of a wider **graph-based approach** for learning. In this approach we first define some graph structure over the samples - forming the nodes of the graph. Then, using the constructed graph we perform training and prediction. The different learners differ in how to define edges in the graph? are they weighted or not? how to transition between nodes? and how is this structure used for training and prediction?

3.5.1 Prediction Using k-NN

Let us begin with the simplest form of k-NN. The first step is to determine two hyper-parameters required by the algorithm: an integer k (the number of neighbors to use) and a distance function $\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$. We can decide for example to use the square Euclidean norm $\rho(\mathbf{x}_1, \mathbf{x}_2) := ||\mathbf{x}_1 - \mathbf{x}_2||^2$ or a weighted square norm giving different importance levels to different features $\rho(\mathbf{x}_1, \mathbf{x}_2) := \sum \omega_i((\mathbf{x}_1)_j - (\mathbf{x}_2)_j)^2$.

Then, given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ the prediction is done as follows:

Algorithm 2 k-NN

procedure k-NN(k, ρ, S,$\mathbf{x}'$)▷ Where k, ρ are the pre-determined hyper-parameters, S the training set and $\mathbf{x}'$ the sample to predict for

Compute distance from $\mathbf{x}'$ with respect to ρ: $\forall \mathbf{x} \in S \quad d_{\mathbf{x}} := \rho(\mathbf{x}', \mathbf{x})$.

Denote $\pi = (\pi_1, \ldots, \pi_m)$ the permutation of $(1, \ldots, m)$ such that $d_{\mathbf{x}_{\pi_1}} \leq \ldots \leq d_{\mathbf{x}_{\pi_m}}$

Select k nearest samples $\mathbf{x}_{\pi_1}, \ldots, \mathbf{x}_{\pi_k}$ and predict by majority vote:

$$\widehat{y} := \underset{y \in \{0,1\}}{argmax} \sum_{i=1}^{k} \mathbb{1}[y_{\pi_i} = y]$$

return $\widehat{y}$

end procedure

3.5.2 Selecting Value of k Hyper-Parameter

A very important aspect in k-NN is the chosen value of k. Though methods for determining the "right" k will be discussed in future chapters, let us dwell on a few cases:

- $k = 1$**:** The test point is given the label of the single nearest neighbor in the training set. Such classifier has a very low bias but very high variances.
- $k = m$**:** The classifier predicts a single label for any given test sample, regardless to its values. It will predict the majority vote of the training set labels. In this case the bias is very high and the variance is zero.

As we change k we change the bias-variance tradeoff, with larger values of k creating simpler models while smaller values of k creating more complex models (Figure 3.12).

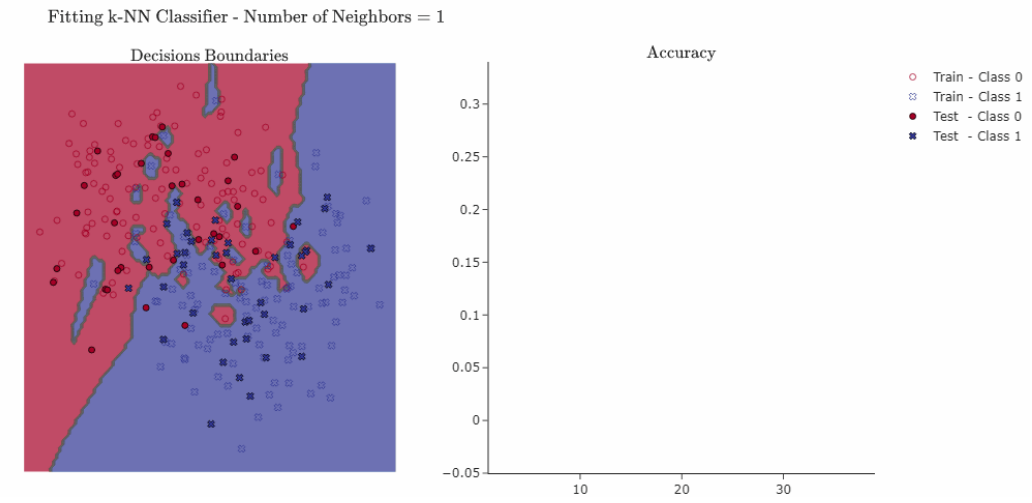


Figure 3.12: Decision Boundaries of k-NN: *Fitting model over dataset for different values of k.* *Chapter 3 Examples - Source Code*

3.5.3 Computational Implementation

Implementing a k-nearest-neighbors classifier is very easy on small datasets, but becomes computationally challenging (either in terms of execution time or in terms of space) when d and/or m are large. There are generally three types of implementation approaches:

- **Brute force implementation:** We keep the entire training sample S in storage during the entire prediction process. For each new test sample $\mathbf{x} \in \mathbb{R}^d$ we calculate $\rho(\mathbf{x}, \mathbf{x}_i)$ $i \in [m]$ and partially sort to find the k smallest distances.
  Suppose ρ is the Euclidean distance. What are the computational costs of prediction? As the sample space is $\mathbb{R}^d$ computing the distance between two points is $\mathcal{O}(d)$. Doing so for all points in the dataset is $\mathcal{O}(dm)$. Next we want to retrieve the k nearest train samples. If $k \ll m$ we can retrieve k times the sample of minimal distance (without repeating previously selected samples) in a time complexity of $\mathcal{O}(km)$. However, if $k \approx ....$ then it is more computationally efficient to sort all distances and then select the k minimal. Lastly, summation over selected samples is done in $\mathcal{O}(k)$. All together the time complexity of such approach is $\mathcal{O}(dm + km)$. In terms of space complexity we must store distances of all training samples and thus $\mathcal{O}(m)$.

- **Exact nearest neighbors search with preprocessed data structure:** Depending on the selection of ρ, we could pre-process the training sample and construct a special data structure. After doing so in the training step, we can use this data structure to quickly locate the k nearest neighbors of a given test sample. In the case of ρ being the Euclidean distance we could you an algorithm such as *kd-tree*.

- **Fast randomized nearest neighbors search:** Beyond the scope of this course.

3.5.4 Learner ID Card

- **Hypothesis class:** This is a "model free" learner and therefore has no hypothesis class
- **Learning principle used for training:** There is no training phase for this learner
- **Computational implementation:** Different strategies for calculating the nearest neighbors. Simplest method is by brute force nearest neighbor search
- **Interpretability:** We do not know why s given sample was predicted as it was. We can only explain near what other training samples it is
- **Family of models:** Indexed according to the hyper-parameter k
- **Storing fitted model:** Must store the entire training sample or some preprocessed data structure
- **Prediction of new sample:** Find the k samples in the training set closest (with respect to the used metric) to the given sample. Predict based on the majority vote of these samples
- **When to use:** When implementation is computationally feasible try this model.

3.6 Decision Trees

Decision Trees are classification and regression methods by which we partition the sample space into disjoint parts. Then, given such a partition, the response of our classification (or regression) is computed based on the training samples in the partition of the observation in question. These methods are very intuitive and yet capture many interesting aspects of learning. We will discuss some of these aspects in details in the following chapter.

R To this day, one of the more powerful classification and regression algorithms is what is called: Classification And Regression Trees (CART) Random Forest. It uses the power of committee decisions over the basic decision trees to achieve very good performances. Some of the aspects of this algorithm will be discussed in later chapters.

3.6.1 Axis-Parallel Partitioning of $\mathbb{R}^d$

Earlier in this chapter we have discussed two classifiers that use piecewise-constant prediction rules: half-spaces and SVM. For both, the hypothesis class consisted of half-spaces where prediction was determined by position of sample with respect to the hyper-plane. For decision trees we will describe a more complicated piecewise-constant prediction rule (more complex hypotheses). Let us consider a rule that partitions the sample space $\mathbb{R}^d$ into **axis-parallel boxes, or "hyper-rectangles"** where each box is associated with labels 1 or -1. The learner's task would be to use the training sample to "chop" the samples space $\mathcal{X}$ int a disjoint union of axis-parallel boxes, and to assign a class prediction to each box.

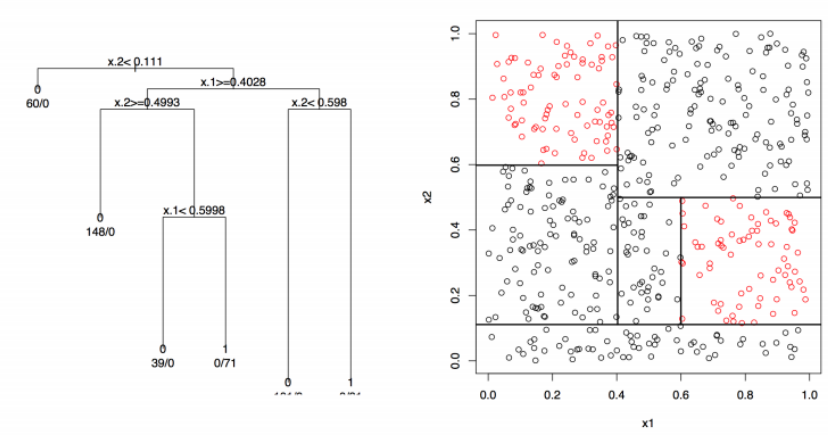


Figure 3.13: *Decision tree and induced partitioning of $\mathbb{R}^2$ sample space*

To make out hypothesis $\mathcal{H}_{CT}$ class smaller and simpler, we will focus on disjoint unions of boxes that are obtained by iteratively splitting an existing box into two smaller boxes along one of the axes:

- We start with the whole sample space $\mathbb{R}^d$.
- By selecting some coordinate $i_1 \in [d]$ and some value $t_1 \in \mathbb{R}$ we split R^d into two axis-parallel "boxes" (half-spaces). We obtain:

$$B_1^+ = \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}_{i_1} > t_1\right\}, \quad B_1^- = \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}_{i_1} \leq t_i\right\}$$

- Next, by focusing of some previously split "box" B_j^s for $s \in \{-,+\}$, we can again select some coordinate $i_{j+1} \in [d]$ and some splitting value $t_{j+1} \in \mathbb{R}^d$ to obtain B_{j+1}^-, B_{j+1}^+. Notice that B_{j+1}^- and B_{j+1}^+ are disjoint, and if following this procedure then by induction they are also disjoint from any other obtained box.

Note that the partitions obtained this way are special - most partitions of $\mathbb{R}^d$ into axis-aligned boxes are not Tree Partitions. Namely, cannot be constructed by such a top-down iterative chopping procedure.

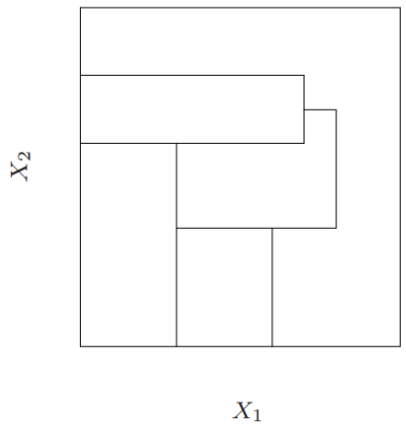


Figure 3.14: *Partitioning $\mathbb{R}^2$ into axis-aligned boxes not describing a tree partition*

3.6.2 Classification & Regression Trees

The hypothesis class $\mathcal{H}_{CT}$ we will consider consists of piecewise-constant functions, that assign a class prediction (1 or 0) to each box in a Tree Partition. Unless we restrict it somehow, the class contains all piecewise-constant functions supported on all Tree Partitions of $\mathbb{R}^d$ (to any number of boxes). Formally, for a Tree Partition $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$ of $\mathbb{R}^d$ into N boxes, and label assignments $c_j \in \{0,1\}$ $(j = 1,\ldots,N)$ assigning label c_j to box B_j, the hypothesis $h \in \mathcal{H}_{CT}$ is a function $h : \mathbb{R}^d \to \{0,1\}$ defined by

$$h(\mathbf{x}) \coloneqq \sum_{j=1}^{N} c_j \mathbb{1}[\mathbf{x} \in B_j]$$

■ **Example 3.4** Consider the following scenario: suppose someone comes into a hispital emergency room. The first step of triage is to determine - fast - whether they are in a life-threatening medical emergency, or else they can wait in line and receive treatment in a little while. The triage uses a sequence of yes/no questions, such as: Is the patient conscious yes/no?

- If not conscious: classify as **emergency**
- If patient is conscious: is the patient's pulse < 40 beats per minute?
  - If yes (pulse < 40): classify as textbfemergency
  - If no, (pulse ≥ 40): is the patient's pulse > 130 beats per minute?
    - If yes (pulse > 130): is the patient's systolic blood pressure < 80 mmHg?
      - If yes classify as **emergency**
      - If not, is the patient's systolic blood pressure > 140 mmHg? If yes, classify as **emergency**. Otherwise, classify as **no emergency**
    - If not classify as **no emergency**

This is a decision tree that uses three features: conscious (a binary categorical feature), pulse (a numerical feature) and blood pressure (also a numerical feature). See if you can we write a diagram for this decision tree in the shape of a tree, where every node is a question, and every leaf is a decision / classification. The root of the tree is the first question ("conscious yes/no?"). Now observe that every function in our Classification Trees hypothesis class $\mathcal{H}_{CT}$ is equivalent to a decision tree.

In the notations of the generic example above, the first question is: "$x_{i_1} > t_1$-yes/no?". If yes, we ask the second question "$x_{i_{2,1}} > t_{2,1}$ - yes/no?". If not, we ask the second question: "$x_{i_{2,2}} > t_{2,2}$ - yes/no?". And so on, until there are no more splits and we have reached a box over which the function in $\mathcal{H}_{CT}$ is constant. If the constant value is 1, we classify / predict class 1. If the constant value is 0, we predict class 0. This is why our hypothesis class is called - classification trees ∎

3.6.3 Growing a Classification Tree

Having defined our hypothesis class, the next question is what learning principle to use. That is, how shall be select $h_s \in \mathcal{H}_{CT}$ based on the training sample S. Suppose we have already obtained a Tree Partition of $\mathbb{R}^d$ is some manner, that consists of N disjoint boxes $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$. Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ be our taining set and denote the predicted label assigned to box B_j by $\hat{y}(B_j) \in \{0,1\}$. As such, the number of misclassification errors that are incurred by the training sample that fall inside B_j is

$$\sum_{\mathbf{x}_i \in B_j} \mathbb{1}\left[y_i \neq \hat{y}(B_j)\right]$$

Let us begin learning by applying the ERM principle. Denote the fraction of misclassified samples with label $y \in \{0,1\}$ in some box B by:

$$\ell_S(B,y) \coloneqq \frac{1}{|B|} \sum_{\mathbf{x}_i \in B} \mathbb{1}\left[y_i \neq y\right]$$

The label that will minimize the empirical risk for those training samples in box B is the **majority vote** over the labels. So, for any sample falling in box B we would predict

$$\hat{y}_S(B) \coloneqq \underset{y \in \{0,1\}}{\operatorname{argmin}} \ell_S(B,y)$$

Applying over the entire Tree Partition, minimizing the empirical risk is achieved by labeling box B_j with $\hat{y}_S(B_j)\,,\ j \in [N]$. Therefore, for a given training set S, every Tree Partition corresponds with a unique label assignment, and as such, a unique classification tree $h \in \mathcal{H}_{CT}$ that minimizes the empirical risk. It seems therefore, that finding the desired ERM tree is done by solving

$$h^* = \underset{h \in \mathcal{H}_{CT}}{argmin}\, L_S(h)$$

where $L_S(h)$ is the misclassification error of h over S. Looking back at the described procedure is it therefore possible to describe which tree would minimize the empirical risk and therefore be selected (for any training set S)? Consider the tree where the number of leaves is $|S|$ and each sample is in a box containing only itself. Following the prediction rule we devised, such a tree would achieve $L_S(h) = 0$. Though we achieved the lowest possible empirical risk, this tree will fail to generalize to new samples. To cope with this problem we should limit the number of levels in the classification tree (equivalent for limiting number of leaves). Denote $\mathcal{H}_{CT}^k$ the hypothesis class of all tree partitions with at most k levels. Now, we will choose k and then using the ERM principle return

$$h^* = \underset{h \in \mathcal{H}_{CT}^k}{argmin}\, L_S(h)$$

Selecting A Value For k:

Note that by adding the hyper-parameter k we now have **a family of hypothesis classes**, one for each value of k. The value of k controls the size of the hypothesis class and therefore controls the bias-variance tradeoff (Figure 3.15):

- For small values of k, the hypothesis class is smaller, containing trees of smaller sizes. Therefore the ERM learner will have a **higher** bias as it can only select simple Tree Partitions. It will also have a **lower** variance: as the boxes are very large, the labels assigned to each box are based on a majority vote of typically many training samples. Therefore changing a few training samples will barely change the selected hypothesis.
- For large values of k, the hypothesis class is much more complex, with more "specialized" trees in it. Therefore the ERM learner will have a **lower** bias and **higher** variance.

Later in the course we will introduce a few methods for selecting the value of k.

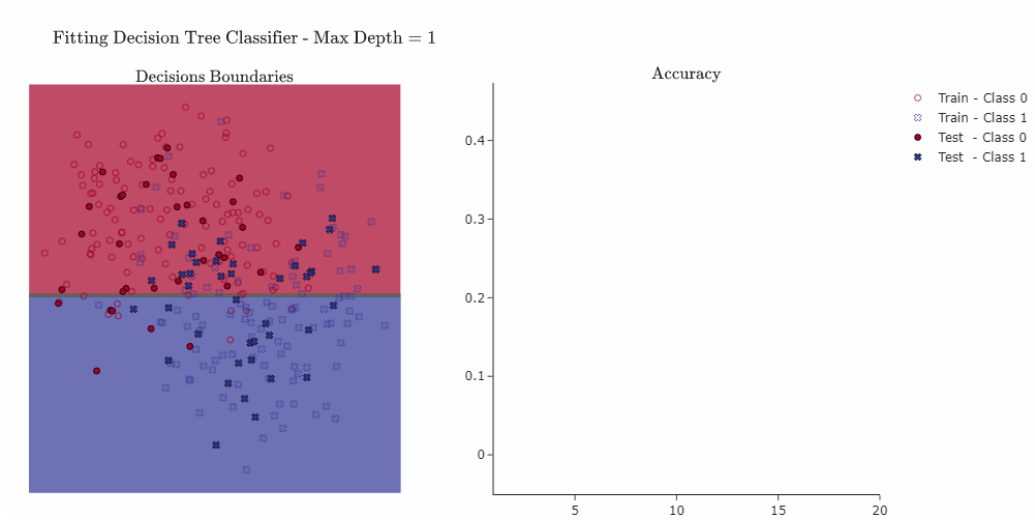


Figure 3.15: Decision Boundaries of Decision Trees: *Fitting model over dataset for different values of max depth k. Chapter 3 Examples - Source Code*

3.6.4 CART Heuristic For Growing Trees

The next challenge is how to find the minimizer of $argmin_{h \in \mathcal{H}^k_{CT}} L_S(h)$ computationally? So far, all the ERM learners we encountered were **computationally tractable**:

- The linear regression optimization problem was based on ERM. We were able to find aclosed form expression for the minimizer.
- The Half-space classifier was based on ERM and lead to a simple convex optimization problem.

In contrast to those examples the search space over $\mathcal{H}^k_{CT}$ is exponentially large and has no Euclidean or other structure to be used. Finding an ERM solution would mean to use brute-force search, which is infeasible. In fact, it has been proven that implementing ERM on $\mathcal{H}^k_{CT}$ is an NP-Hard problem with respect to the training sample size[1] .

[1] By reduction from "three dimensional matching", see Hyafil and Rovest, "Constructing Optimal Binary Decision Trees is NP-Complete", Information Processing Letters 5(1), 1976

This is our first encounter with the bitter truth that though the ERM principle is nice, it is often impossible to implement efficiently, especially when the hypothesis class has no Euclidean structure. Therefore, we must resort to defining and using **heuristics**: an approach to solving the optimization problem that does not guaranteed to be optimal, but is still sufficient for finding a solution. While the definition of decision trees, the hypothesis class and prediction assignment to boxes in a tree partition are all canonical, there are several different heuristic approaches to the way we "grow a decision tree', namely to the way we choose an hypothesis in practice.

One common approach, coming out of the statistical learning community, is called **Classification and Regression Trees** (CART). This heuristic cinsists of two stages: **growing** the tree, resulting in a tree that is a little too large, and then **pruning** it to bring it down to the most effective size.

Suppose we have chosen k to be the maximal tree depth. The heuristic of growing a full decision tree with at most k levels will proceed top-down, starting from $\mathbb{R}^d$ and progressively chopping each box into two boxes. A given box is **not chopped** if either:

- The maximum number of levels k has been reached.
- The box has reached a pre-determined minimal number of training samples. At the very least we would not split a box if it consists of only a single training sample.

Chopping is done by finding the **best** coordinate, at the **best** value to chop, and whenever you chop given each half-box the **best** class assignment, in the sense of minimizing misclassification error over the training sample. Formally, the pseudocode of the CART heuristic is seen in Algorithm 3.

Time Complexity Analysis

Before we introduced the CART heuristic we described that solving the ERM principle over this hypothesis class is NP-Complete and therefore cannot be done in polynomial time. Let us see that the CAR heuristic can indeed be computed efficiently.

The algorithm iteratively splits boxes into two by finding a coordinate $i \in [d]$ and a value $t \in \mathbb{R}$. It scans all d coordinates and for each scans all possible values of t. Notice, that even though $t \in \mathbb{R}$ we do not need to try all values and it suffices to check only the values in the i'th coordinate of the training sample: $\{\mathbf{x}_i | \mathbf{x} \in S\}$. As we have m samples we only need to evaluate for at most m values, giving each step a time complexity of $\mathcal{O}(md)$.

The next question is how many steps will the algorithm perform? We know that the algorithm will terminate after growing a tree with at most k levels. Such a tree will have at most $2^k - 1$ nodes and leaves. Though this seems exponentinal in the given input (notice that the hyper-parameter k is also part of the input) we can upper bound this value. As we do not allow empty boxes (and in fact any box with less than some minimal number of samples) the number of nodes (including leaves) in the tree is at most m. We therefore conclude that the time complexity of the CART heuristic is $\mathcal{O}\left(m^2 d\right)$

Pruning a Decision Tree

Pruning a tree means cutting off unnecessary branches. The tree obtained when we are done with the "growing" stage of CART may be too large. A tree too large means some of the boxes are too small, so we are not in an optimal point on the bias-variance tradeoff. It could help reduce the generalization error to merge some of the boxes together, so that the majority votes to determine

Algorithm 3 CART - For Growing Classification Tree

procedure CART($S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, k, m_{min})
  $Tree - Partition \leftarrow \emptyset$
  $Boxes \leftarrow \{\mathbb{R}^d\}$ ▷ Entire sample space as initial box
  while $Boxes \neq \emptyset$ **do**
    $B \leftarrow \text{pop}(Boxes)$
    if $|B| \leq m_{min}$ or depth at B reached k **then**
      $Tree - Partition \leftarrow Tree - Partition \cup \{B\}$
      continue
    end if
    for all feature $i \in [d]$ **do** ▷ Scan all features
      for all threshold value $t \in \mathbb{R}$ **do** ▷ Scan thresholds for features
        Split B along coordinate i at value t:

$$B_{i,t}^+ := \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}_i > t\right\}, \quad B_{i,t}^- := \left\{\mathbf{x} \in \mathbb{R}^d | \mathbf{x}_i \leq t\right\}$$

        Let $\hat{y}(B_{i,t}^{\pm})$ denote the class assignment for boxes $B_{i,t}^{\pm}$.
        Let $\ell_S(B_{i,t}^{\pm}, \hat{y}(B_{i,t}^{\pm}))$ denote the empirical risk incurred by $\hat{y}(B_{i,t}^{\pm})$.
        Define $g_i(t) := \ell_S(B_{i,t}^+, \hat{y}(B_{i,t}^+)) + \ell_S(B_{i,t}^-, \hat{y}(B_{i,t}^-))$
      end for
      Set the **best** splitting point: $t_i \leftarrow \text{argmin}_{t \in \mathbb{R}}\, g_i(t)$
    end for
    Select **best** feature to split by: $i^* \leftarrow \text{argmin}_{i \in [d]}\, g_i(t_i)$
    $Boxes \leftarrow Boxes \cup \left\{B_{i^*,t_{i^*}}^+, B_{i^*,t_{i^*}}^-\right\}$ ▷ Split box by best feature and threshold
  end while
  return $Boxes$
end procedure

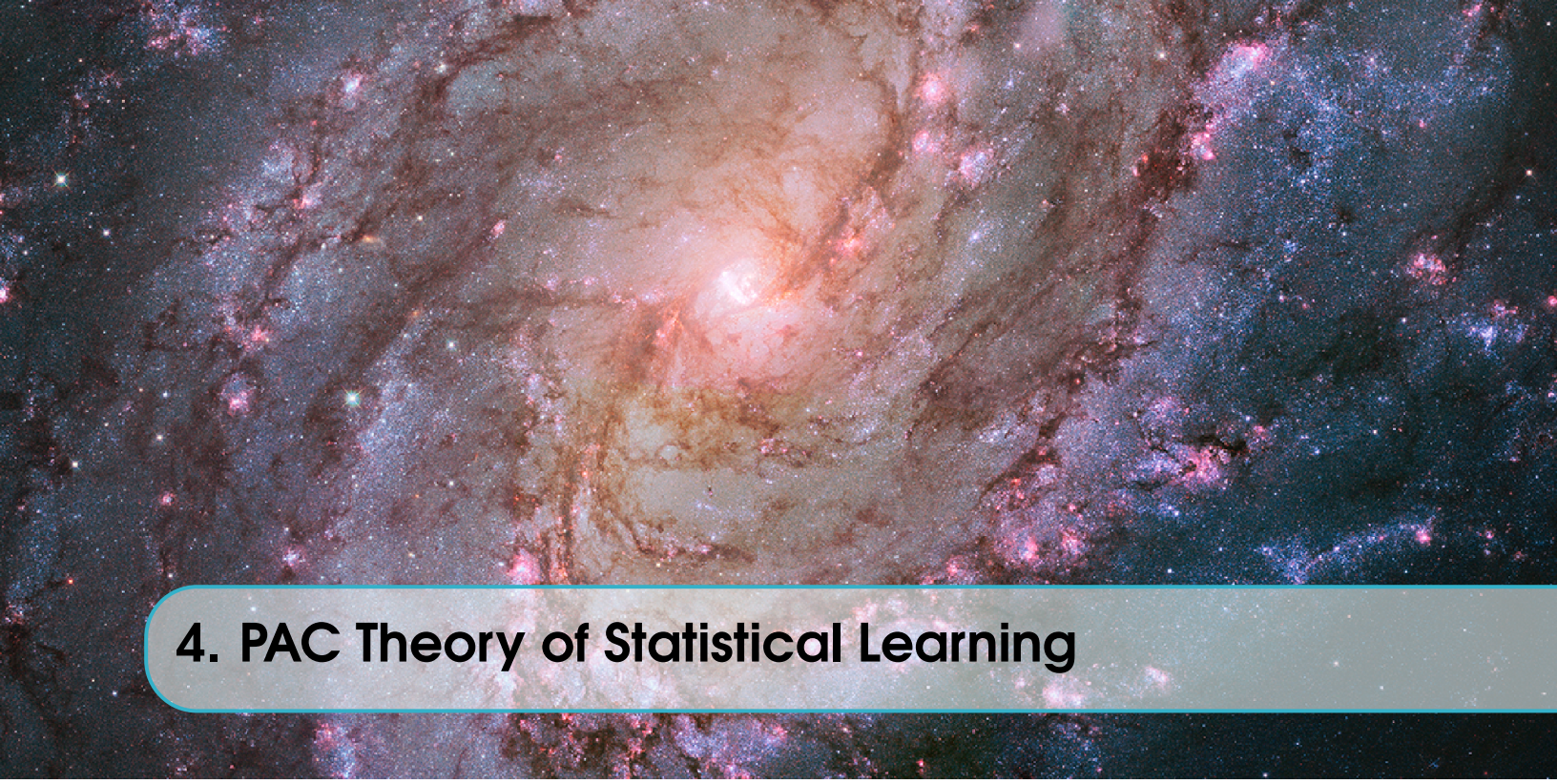
the box label assignment would be based on larger sets of training samples. Merging two boxes is equivalent, from the decision tree perspective, to merging to leaves together and removing the node between them. Hence, "pruning". We will complete the CART heuristic when we discuss regularization (6.1.1).

3.6.5 Interpretability

One of the great advantages of a classification tree is that it is very interpretable. To understand which features where important in the classification process, we just look at the nodes (the splits) and see which features the classification tree algorithm chose to split on. A feature that never appeared in any split has not been useful for classification of the training sample. A feature that appears once or more (remember that the algorithm can choose to split on some feature again and again in different areas of R^d) has been useful. To understand why a new sample was classified the way it was classified, we just follow the tree from top to bottom, and see how each answer to each question went.

3.6.6 Learner ID Card

- **Hypothesis class:** The piecewise-constant functions induced by Tree Partitions (axis-aligned rectangles) of depth at most k: $\mathcal{H}_{CT}^{k}$
- **Learning principle used for training:** ERM
- **Computational implementation:** Implementation of ERM is NP-Hard. Therefore we use heuristics such as the top-down greedy heuristic of CART
- **Interpretability:** A very interpretable learner. Simply reading the tree structure
- **Family of models:** Indexed by k the maximal tree depth
- **Storing fitted model:** To store this model we must store for each node the split information: the coordinate i by which to split and the threshold value t. In addition we need to store the label assignment at the leafs of the tree
- **Prediction of new sample:** Navigate top-down along the tree until reaching a leaf
- **When to use:** This classifier is used as a simple baseline or to get a highly interpretable rule that is easy to explain and plot. Otherwise, classification trees are used to construct "random forests" which are covered in 5.3.4



4. PAC Theory of Statistical Learning

4.1 A Theoretical framework for learning

The basic questions in machine learning are: Which tasks are learnable? How do we learn learnable tasks? How many training samples do we need in order to learn them? In this chapter we develop the ***PAC theory of learnability***, which provides us, within its definitions and assumptions, a complete answer to these questions, for ***batch supervised learning***.

A Data-generation Model

The two basic assumptions in the PAC framework are:

- There exists a (deterministic) function f which is the correct classifier, i.e., for every $\mathbf{x}$ there is a single correct label, given by $y = f(\mathbf{x})$. We shall refer to this case as the ***PAC Model***.

- All samples, either in the training set or in any future test set are independent and identically distributed (i.i.d) random variables, i.e., they are sampled independently using a distribution $\mathcal{D}$ over the sample space $\mathcal{X}$. In particular, this means that the probability, $\mathbb{P}(S)$, of drawing the sequence $\mathbf{x}_1, \ldots, \mathbf{x}_m$ (which equals, due to the previous assumption, the probability of getting the sequence $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ with $y_i = f(\mathbf{x}_i)$) is given by $\mathbb{P}(S) = \prod_{i=1}^m \mathcal{D}(\mathbf{x}_i)$.

R Later on, in section 4.4, we shall relax these assumptions and consider a more general case which we will call the ***Agnostic PAC model*** and in which the *same* $\mathbf{x}$ may appear with *different* labels, meaning that the labels themselves will be random, at least to some extent, and therefore also the probability density will be defined over $\mathcal{X} \times \mathcal{Y}$.

Let us compare the assumptions of the PAC model to the linear regression model covered in chapter 2. In both cases we have received a set of examples $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$. In the case of linear regression we implicitly assumed all the $\mathbf{x}_i$'s had equal importance, for example, in their contribution to the

global error. Now, the $\mathbf{x}_i$'s are *i.i.d* sampled according to $\mathcal{D}$, i.e., they have different probabilities to appear and therefore may have different weights in the loss function. Another difference is that in the case of the linear model we started with the assumption $y_i = f(\mathbf{x}_i)$ where f was deterministic and linear. Here, we do assume f to be deterministic, but otherwise it may take the form of any possible function from $\mathcal{X}$ to $\mathcal{Y}$.

In the linear model we eventually relaxed the deterministic assumption and considered y to be a random function of $\mathbf{x}$ of the form: $y_i = f(\mathbf{x}_i) + z_i$. An analogous generalization will take place also here, once we consider the more general, agnostic, case. Finally we note that, although this chapter will focus on *classifiers*, many principles we will encounter will hold also for linear regression problems.

Generalization Error for Classifiers

For a classification task, we define the ***Generalization Error*** of a hypothesis h as the probability to obtain an $\mathbf{x}$ for which $h(\mathbf{x})$ is different than the correct label $f(\mathbf{x})$:

$$L_{\mathcal{D},f}(h) \equiv \mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[h(\mathbf{x}) \neq f(\mathbf{x})] \equiv \mathcal{D}(\{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) \neq f(\mathbf{x})\}) \tag{4.1}$$

where $\mathcal{D}$ and f are unknowns. The generalization error is also called the ***Risk***, or the ***True Error***. Note, one should be critical about an error measure that counts the total number of misclassification errors of a classifier. Recall the distinction we make between the Type-I and Type-II errors, where often the one is worse than the other. For now however, we only consider the generalization error with respect to the misclassification error and therefore do not distinguish between the two types of errors.

The Fundamental Theorem of Statistical Learning

So our task is to design a learning algorithm (a learner), $\mathcal{A}$. The algorithm will receive a training sample of size m $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ and outputs a prediction rule (a hypothesis) $h : \mathcal{X} \to \mathcal{Y}$. For classification problems, for example, $\mathcal{Y} = \{\pm 1\}$, but the framework we develop has a much broader applicability.

We assume that the data points, both in training set and in the test set, are generated by sampling independently $\mathcal{X}$ using a distribution $\mathcal{D}$, which is unknown to us. The labels y are fixed: given a particular $\mathbf{x}$ there exists some deterministic function f such that $y = f(\mathbf{x})$, where f is unknown to us and the training set is our only view to what f does.

Finally, the performance of any candidate rule $h : \mathcal{X} \to \mathcal{Y}$ that our learner may produce, will be evaluated by how well it will perform on future, unseen samples. This is done using the expected misclassification rate $L_{\mathcal{D},f}(h) \equiv \mathbb{P}_{\mathbf{x}\sim\mathcal{D}}[h(\mathbf{x}) \neq f(\mathbf{x})]$. The next sections will be devoted for a detailed understanding of the following important definitions:

Definition 4.1.1 A hypothesis class, $\mathcal{H}$, is a ***PAC Learnable hypothesis class*** if there exist a learning algorithm $\mathcal{A}$ and a function $m_{\mathcal{H},\mathcal{A}} : (0,1)^2 \to \mathbb{N}$ with the following property:

- For every $\varepsilon, \delta \in (0,1)$
- For every distribution $\mathcal{D}$ over $\mathcal{X}$
- For every labeling function $f : \mathcal{X} \to \{\pm 1\}$ that such that there exists $h^* \in \mathcal{H}$ which satisfies $L_{\mathcal{D},f}(h^*) = 0$

when running the learning algorithm $\mathcal{A}$ on $m \geq m_{\mathcal{H},\mathcal{A}}(\varepsilon,\delta)$ *i.i.d* samples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis $h_S = \mathcal{A}(S)$ such that, with probability of at least $1-\delta$ (over the choice of the training samples), we have $L_{\mathcal{D},f}(h_S) \leq \varepsilon$.

$$\mathcal{D}^m\left(S \middle| L_{\mathcal{D},f}(h_S) \leq \varepsilon\right) \geq 1-\delta$$

Denote the minimal sample size required for the above conditions to hold with respect to ε,δ and with respect to any algorithm, by $m_{\mathcal{H}}(\varepsilon,\delta) = \min_{\mathcal{A}} m_{\mathcal{H},\mathcal{A}}(\varepsilon,\delta)$. The function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ is called the ***Sample Complexity*** of the PAC learnable hypothesis class $\mathcal{H}$.

Definition 4.1.2 Let $\mathcal{H} \subseteq \{\pm 1\}^{\mathcal{X}}$ be a hypothesis class. For a subset $C \subset \mathcal{X}$ let $\mathcal{H}_C$ be the restriction of $\mathcal{H}$ to C, namely, $\mathcal{H}_C = \{h_C : h \in \mathcal{H}\}$, where for $h : \mathcal{X} \to \mathcal{Y}$, $h_C : C \to \mathcal{Y}$ is the function such that $h_C(\mathbf{x}) = h(\mathbf{x})$ for every $\mathbf{x} \in C$. Define the ***VC-dimension*** of $\mathcal{H}$ by:

$$VCdim(\mathcal{H}) = \max\left\{|C| \;\middle|\; C \subset \mathcal{X} \text{ and } |\mathcal{H}_C| = 2^{|C|}\right\}$$

By choosing PAC learnability as our interpretation of what learnability means, definitions Definition 4.1.1 and Definition 4.1.2 provide a *well defined necessary and sufficient condition* for when learning is possible and the minimal training sample size needed in order to learn. This framework also provides a "universal" learner that successfully learns given a sufficiently large training set. In short, we have a full theory of batch learning - a full theory of when it is possible to generalize from a training sample to new samples, and how to do it. This result is sometimes known as "***The Fundamental Theorem of Statistical Learning***" and states that:

- A hypothesis class $\mathcal{H}$ is PAC-learnable *if and only if* $VCDim(\mathcal{H})$ is finite.
- The sample complexity of a hypothesis class with a finite VC-dimension is given approximately by
$$m_{\mathcal{H}}(\varepsilon,\delta) \sim \frac{VCDim(\mathcal{H}) + \log(1/\delta)}{\varepsilon}$$
- The ERM rule achieves this minimum, namely, when learning is possible, ERM learns with a minimial number of examples.

The first step in gaining a detailed understanding of PAC-learnability, VC-dimension and the fundamental theorem, will be to present a different perspective of the above framework.

4.1.1 Learning As A Game - First Attempt

The framework in section 4.1 can be thought of as a *game* between us and Nature, with a random payoff. The game proceeds as follows. First, the number of training samples m is determined in advanced as a game parameter.

We perform the first step and choose some learner $\mathcal{A}$ that trains on m examples. This is our strategy. Then Nature makes the second step and chooses a distribution $\mathcal{D}$ and a labeling function f. This is Nature's strategy. Importantly, Nature knows the strategy we chose when she chooses her strategy. To calculate the game's payoff, an *i.i.d* sample S of size m is drawn according to the distribution $\mathcal{D}$

and labeled according to the function f, both of which were chosen by Nature. This set is then fed into the learner $\mathcal{A}$ that we chose to obtain the prediction rule $h_S = \mathcal{A}(S)$. The notation h_S emphasizes that the prediction rule we learn, h_S, strongly depends on the random sample S. The payoff is $L_{\mathcal{D},f}(h_S)$.

Our goal in the game is to end up with an $L_{\mathcal{D},f}(h_S)$ which is as small as possible while Nature is an adversary that wants $L_{\mathcal{D},f}$ to be as large as possible. Note that the payoff is random as it depends on the sample S that was drawn. If we are unlucky, and the sample S is "bad", namely, does not represent $\mathcal{D}$ very well then the rule h_S our learner produces might not generalize well and the random loss (for that draw of S) will be high.

Under such setting, what is the best strategy for us? How to best design the learner $\mathcal{A}$? Remember that Nature will know what we chose, and can try to be "cruel", namely, to choose $\mathcal{D}$ and f that our learner did not prepare well for. Also, there is always a chance that we will draw a "bad" sample S, which will mislead us in choosing a rule h_S that will not generalize well. So, is there a way to defend ourselves against "cruel" strategies $\mathcal{D}, f$ that Nature might play, and against unlucky draws of a training sample S? Is there anything at all that can be said in this generality about the problem of learning (i.e., the problem of generalizing from training samples to new samples)?

Definition 4.1.3 ***The Learning Game*** (first version):

- A sample size m is fixed.
- We choose a strategy (a learner) $\mathcal{A}$. That is, a function that matches every sample S to a prediction rule $h_S : \mathcal{X} \to \mathcal{Y}$.
- Nature knows our strategy and, after us, chooses a strategy that consists of a probability distribution $\mathcal{D}$ over $\mathcal{X}$, and a label function $f : \mathcal{X} \to \mathcal{Y}$.
- A sample S of size m is drawn according to $\mathcal{D}$ and is labeled according to f.
- The sample S is fed into $\mathcal{A}$ to produce a prediction rule $h_S = \mathcal{A}(S)$.
- The payoff is $L_{\mathcal{D},f}(h_S)$, namely, the expected fraction of misclassification errors h_S will make on future data drawn $i.i.d$ according to $\mathcal{D}$ and labeled according to f. The payoff is *random* since S is random and therefore h_S is random.
- Nature does her best to win. So we will look for learners $\mathcal{A}$ for which we can *ensure* that the loss $L_{\mathcal{D},f}(h_S)$ ever exceeds a certain value, *for any* strategy $\mathcal{D}, f$ that Nature might play.

4.1.2 Probably Correct & Approximately Correct Learners

The generalization loss, $L_{\mathcal{D},f}(h_S)$, is random since it depends on the randomly-drawn training sample, S. Therefore, in general, one should talk about the *probability* that a learner has a certain loss. In particular, saying that $L_{\mathcal{D},f}(h_S)$ *never* exceeds a certain value, actually means that (for the specific f, $\mathcal{D}$ and $\mathcal{A}$) the probability of drawing a training sample S that, for which, $\mathcal{A}$ produces a rule with a loss smaller than a certain value, is 1. This brings us to the following definition:

Definition 4.1.4 Let $\varepsilon \in (0,1)$. We say that a learner $\mathcal{A}$ is ***Approximately Correct*** with an ***accuracy*** ε, if we are certain (with probability 1) that for any training sample, S, drawn using $\mathcal{D}$, $\mathcal{A}$ will output a prediction rule, h_S, with a loss smaller or equal to ε:

$$\mathcal{D}\left(S \middle| L_{\mathcal{D},f}(h_S) \leq \varepsilon\right) = 1$$

Is there an accuracy parameter, $\varepsilon \in (0,1)$, for which we can find an approximately correct learner? Unfortunately, the answer to this question is no. For any ε, Nature always has a strategy that can make sure that there is a non-zero probability, even if very small, to get a "pathological" training sample S that does not represent $\mathcal{D}$ at all. The resulting rule h_S can be wrong on most of $\mathcal{X}$, which would cause a loss arbitrarily close to 1, higher than any specified ε. We shall prove this assertion by giving an example of one such strategy that Nature can use.

■ **Example 4.1** Let $\varepsilon \in (0,1)$ and consider any two points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Let us denote by S' a training sample that happened to have only $\mathbf{x}'$'s in it (in particular, S' does not contain $\mathbf{x}$): $S' = (\mathbf{x}_1, y_1), ..(\mathbf{x}_m, y_m)$ with $\mathbf{x}_i = \mathbf{x}'$ and $y_i = f(\mathbf{x}')$ for $i = 1..m$. Although such a sample might be rare, it can not be excluded and therefore, our algorithm, $\mathcal{A}$, should specify what label its output rule, $h_{S'}$ should predict on other points beside $\mathbf{x}'$, and in particular on $\mathbf{x}$, in case it receives S' as an input. Assume, without a loss of generality, that we choose $h_{S'}(\mathbf{x}) = +1$, that is, if the training sample is S' then $\mathcal{A}$ will predict $+1$ on $\mathbf{x}$.

Nature's strategy comprises of two parts. First, it reduces $\mathcal{X}$ to a 2-point space by choosing $\mathcal{D}$ that vanishes everywhere except on $\mathbf{x}$ and $\mathbf{x}'$. More specifically, it chooses $\mathcal{D}$ with $\mathcal{D}(\mathbf{x}) = \gamma$, $\mathcal{D}(\mathbf{x}') = 1 - \gamma$, where γ is a number satisfying $0 < \varepsilon < \gamma < 1$. Second, Nature chooses a labeling function f with $f(\mathbf{x}) = -1 = -h_{S'}(\mathbf{x})$. The probability of obtaining S' is not zero since it is given by $(1-\gamma)^m > 0$. Therefore, to show that Nature's strategy prevents $\mathcal{A}$ from being approximately correct with an accuracy ε, all we need to show is that the loss in the case of obtaining the set S', is larger than ε. Indeed:

$$L_{\mathcal{D},f}(h_{S'}) = \mathcal{D}(\mathbf{x}) \mathbb{1}[f(\mathbf{x}) \neq h_{S'}(\mathbf{x})] + \mathcal{D}(\mathbf{x}') \mathbb{1}[f(\mathbf{x}') \neq h_{S'}(\mathbf{x}')]$$

where $\mathbb{1}[a \neq b]$ is 1 if $a \neq b$ and 0 otherwise. Both terms on the right are non-negative and therefore:

$$L_{\mathcal{D},f}(h_{S'}) \geq \mathcal{D}(\mathbf{x}) \mathbb{1}[f(\mathbf{x}) \neq h_{S'}(\mathbf{x})] = \gamma \mathbb{1}[-1 = -1] = \gamma > \varepsilon$$

So, for any fixed $\varepsilon \in (0,1)$, for any strategy $\mathcal{A}$ we might play, Nature has a strategy $\mathcal{D}, f$, such that there is a non-zero probability over the choice of training samples of length m, that we will have $L_{\mathcal{D},f}(h_S) > \varepsilon$. As ε was arbitrary, that means that Nature can, with a non-vanishing probability, cause the game to end with a loss arbitrarily close to 1. ■

In the above example, even if very small, with probability $(1-\gamma)^m$ we get a "bad" training sample, S, that does not represent $\mathcal{D}$ good enough, and does not allow us to generalize. We therefore conclude, that given any accuracy parameter $\varepsilon \in (0,1)$, no learner $\mathcal{A}$ can guarantee, that with probability 1, the loss will not exceed ε. Nature can always find a strategy for which $L_{\mathcal{D},f}(h_S) > \varepsilon$ will have a non-zero probability to occur.

So the possibility of bad samples means that we should not aspire for an *absolute* certainty in achieving a limited loss. We will accept the fact that on such bad training samples, the learner $\mathcal{A}$ might fail completely (i.e., produce h_S with a potentially high loss). We will therefore, only require a *limited* certainty, that we will call *confidence*, for having a limited loss. This means that we will demand that the probability of drawing a bad sample will not exceed a certain threshold, $\delta \in (0,1)$. This parameter too will be specified prior to the beginning of the game, together with ε.

Since we no longer demand absolute confidence in having a limited loss, perhaps we can instead require absolute accuracy (zero loss), but with a limited confidence? That is, perhaps we can require that at least for those good training samples (the ones that will have a probability of at least $1-\delta$ to appear) the loss will vanish?

Definition 4.1.5 Let $\delta \in (0,1)$. We say that a learner, $\mathcal{A}$, is ***Probably Correct*** with a ***confidence*** δ, if the probability to obtain a training sample, S, for which $\mathcal{A}$ will output a prediction rule, h_S, with a perfect accuracy (zero loss), is larger or equal to $1-\delta$:

$$\mathcal{D}^m\left(S|\, L_{\mathcal{D},f}\left(h_S\right)=0\right) \geq 1-\delta$$

Note that in definition 4.1.4 we required prefect confidence $\delta = 0$ (i.e., with probability 1) to have a certain accuracy ($0 < \varepsilon < 1$), while in definition 4.1.5 we require a certain confidence $0 < \delta < 1$ to have a prefect accuracy $\varepsilon = 0$. In addition, although it is named "confidence", δ actually means *lack of confidence* since the larger it is, the less sure we can be that our loss is limited.

Is there a confidence parameter $\delta \in (0,1)$, for which we can find a probably correct learner? Here too, the answer is no. For any δ, Nature always has a strategy that with probability larger than $1-\delta$ will cause the rule to have a non-vanishing loss: $L_{\mathcal{D},f}\left(h_S\right) > 0$.

Recall that zero loss means that, with probability 1 with respect to $\mathcal{D}$, the predicted label is always correct. Nature can play a $\mathcal{D}$ that gives a finite but tiny probability to a certain $\mathbf{x} \in \mathcal{X}$. Then, with high probability, the training sample will not include $\mathbf{x}$, and therefore, whatever label the learner assigns to $\mathbf{x}$, it will be a guess and therefore might be incorrect, causing a finite loss. Again, we shall prove this assertion through an example of such a strategy that Nature may adopt.

■ **Example 4.2** Let $\delta \in (0,1)$ and consider any two points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Similar to 4.1, we consider the training sample $S' = (\mathbf{x}_1, y_1), ..(\mathbf{x}_m, y_m)$ with $\mathbf{x}_i = \mathbf{x}'$ and $y_i = f(\mathbf{x}')$ for $i = 1..m$, and assume that once this sample is fed into our algorithm, it will predict $h_{S'}(\mathbf{x}) = +1$.

As before, Nature's strategy is to chooses a labeling function f with $f(\mathbf{x}) = -1 = -h_{S'}(\mathbf{x})$ and a distribution, $\mathcal{D}$, with $\mathcal{D}(\mathbf{x}) = \gamma, \mathcal{D}(\mathbf{x}') = 1-\gamma$. This time however, γ is chosen such that $\delta < (1-\gamma)^m$. This means that when m is large, the probability of getting $\mathbf{x}$, namely, γ, is chosen to be very small. Therefore, in th present case S' is a typical sample, in the sense that its probability to appear, $(1-\gamma)^m$ is larger than δ.

The same calculation as in example 4.1 yields $L_{\mathcal{D},f}\left(h_{S'}\right) \geq \gamma$ and since $\gamma > 0$ that would mean a non-zero loss. Since the probability of obtaining S' is larger than δ, the probability of ending up with a non-zero loss is also larger than δ. In other words, our learner $\mathcal{A}$ is not a probably correct one. ■

We conclude that for any confidence parameter $\delta \in (0,1)$, no learner $\mathcal{A}$ can guarantee, that with probability of at least $1-\delta$, the loss will vanish: Nature can always find a strategy for which $L_{\mathcal{D},f}\left(h_S\right) > 0$ will have a probability large than δ to occur. Even a "typical" training sample may miss small areas in $\mathcal{X}$. On these areas the resulting h_S might be wrong.

Examples 4.1 and 4.2 teach us that no matter what learner we construct, we can never have an absolute confidence that our loss will be limited, neither we can have a limited-confidence that we can obtain an absolute accuracy. What we *might* be able is to have *some* confidence that our loss will not exceed *some* threshold. This brings us to the following definition.

Definition 4.1.6 Let $\delta, \varepsilon \in (0,1)$. We say that a learner, $\mathcal{A}$, is ***Probably Approximately Correct*** with a confidence δ and an accuracy ε if and only if the probability of obtaining a training sample S, for which $\mathcal{A}$ will output a prediction rule, h_S, with a loss that does not exceeds ε, is larger or equal to $1-\delta$:

$$\mathcal{D}^m\left(S|L_{\mathcal{D},f}\left(h_S\right) \leq \varepsilon\right) \geq 1-\delta$$

It is important to understand the difference between the *accuracy* ε and the *confidence* δ.

- Recall that we first draw the training sample S at random. Then, the learning algorithm runs on this random sample and its prediction is therefore random. If S is by chance "weird" (not representing $\mathcal{D}$ well), the rule h_S produced will be "wrong", namely, it will not generalize well. The number δ is the probability of failure due to a "weird" sample S.
- After the learner outputs a rule h_S, it is then tested on a new sample. The new sample is also random. $L_{\mathcal{D},f}\left(h_S\right)$ is the expected fraction of errors h_S will make, i.e., its accuracy, over such data. The number ε refers to that accuracy.

4.1.3 Learning As A Game - Second Attempt

Since we can only hope to build a *Probably Approximately* correct learner, we will update the game definition. The sample size m will no longer be fixed. Instead, the accuracy ε and confidence δ, which our learner is required to achieve, are specified as game parameters. We get to decide on m as part of our strategy. Note that there is a subtle nuance in notation now. When we write $\mathcal{A}$ for the learner, we actually mean a *sequence* of learners - one for each sample size m. It would be better to write $\mathcal{A}_m : (\mathcal{X} \times \mathcal{Y})^m \to \mathcal{Y}^{\mathcal{X}}$. However, to keep the notation simple we will continue writing $\mathcal{A}$, and understand that there is in fact a dependence also on m.

Definition 4.1.7 ***The Learning Game*** (second version):
Fix the desired accuracy and confidence parameters $\varepsilon, \delta \in (0,1)$ and then:

- We choose a sample size m and a learner $\mathcal{A}$, both of which may depend on (ε, δ).
- Nature knows our strategy, and, after us, chooses a strategy that consists of a probability distribution $\mathcal{D}$ over $\mathcal{X}$, and a label function $f : \mathcal{X} \to \mathcal{Y}$. Nature's strategy may too depend on the specified (ε, δ) and also on our choice of m and $\mathcal{A}$.
- A sample S of size m is drawn according to $\mathcal{D}$ and is labeled according to f.
- The sample S is fed into $\mathcal{A}$ to produce a prediction rule h_S.
- The payoff is $L_{\mathcal{D},f}\left(h_S\right)$. It is random since S is random and therefore h_S is random.
- Nature does her best to win. So we will look for learners $\mathcal{A}$ for which there is a probability of at least $1-\delta$ that the loss $L_{\mathcal{D},f}\left(h_S\right)$ will not exceed ε, no matter what strategy $\mathcal{D}, f$ Nature might play.
- To determine if we were successful in the game, we play the game many many times. Each time both us and Nature play the same strategies. Each time however, the training samples drawn are different. calculate the probability, over the random draws of training samples S,

of the event $\left\{S \sim \mathcal{D}^m \,\middle|\, L_{\mathcal{D},f}(h_S) \leq \varepsilon\right\}$. If this probability is found to be larger than $1-\delta$, that is, if the learner $\mathcal{A}$ we chose was Probably Approximately correct with accuracy ε and confidence δ - against Nature's best strategy - we say that *we have been successful (with regards to the parameters ε, δ).*

The game perspective, although phrased differently, is equivalent to the more standard description of our learning challenge, which is the following: The learner doesn't know $\mathcal{D}$ and f. The learner receives an accuracy parameter ε and a confidence parameter δ. It then asks for training data, S, containing $m(\varepsilon, \delta)$ examples (that is, the number of examples can depend on the value of ε and δ, but it can not depend on the unknown $\mathcal{D}$ or f). Finally, the learner should output a hypothesis h_S, that depends only on ε, δ and the training sample S drawn, such that with probability of at least $1-\delta$ it holds that $L_{\mathcal{D},f}(h_S) \leq \varepsilon$. That is, the learner should be Probably Approximately correct, with the specified accuracy ε and confidence δ.

Since we can now choose the sample size m and since data costs time and money, we want m to be as small as possible though we should expect a certain trade-off between ε, δ and our choice of m.

4.2 No Free Lunch and Hypothesis Classes

It turns out that, unfortunately, we cannot, in general, be successful against Nature even in the second version of our game. With no restrictions placed on the choice of $\mathcal{D}$ or f, we can not be confident-enough that we can find an accurate-enough h_S. This is true no matter how large is our sample size m.

In examples 4.1 and 4.2 above, Nature used $\mathcal{D}$ that vanishes everywhere in the sample space $\mathcal{X}$ except for two points. In particular, Nature's strategies were enough to prevent us from constructing a learner that enjoys confidence $\delta = 0$ or accuracy $\varepsilon = 0$ for any $\mathcal{X}$ with two or more points. In the following example, Nature's strategy can be applied only if $\mathcal{X}$ has an infinite number of points (though it is enough to have a countable infinite amount of them).

■ **Example 4.3** Suppose that $|\mathcal{X}| = \infty$ and follow the steps of the second version of the game for some $\delta \in (0,1)$ and some $0 < \varepsilon < \frac{1}{2}$.

- We fix m.
- We now decide what label to predict on a point that does not appear in the training sample, S. We could, for example, decide that whenever a point, say, $\mathbf{x}_4$, does not appear in S but $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ do appear, all with the label +1, then we take $h_S(\mathbf{x}_4) = 1$ but if these 3 points, all show up with the label -1, then we take $h_S(\mathbf{x}_4) = -1$. That would mean that the value we choose for $h_S(\mathbf{x}_4)$ would depend on S. However, we will restrict ourselves to choosing the *same* value, $h_S(\mathbf{x}_4)$, whenever S does not contain $\mathbf{x}_4$, independently of which specific S it is. In other words, we choose a function, $g(\mathbf{x})$, and if a point $\mathbf{x} \in \mathcal{X}$ is *not* observed in S, then at this point, the rule, $h_S(\mathbf{x})$ that $\mathcal{A}$ outputs, will satisfy $h_S(\mathbf{x}) = g(\mathbf{x})$, independently of the specific S we got.
- Nature knows m, so it picks some finite set $C \subset \mathcal{X}$ with $|C| > 2m$, and chooses $\mathcal{D}$ to vanish outside C while being uniform over it: for any $\mathbf{x} \in C$, $\mathcal{D}(\mathbf{x}) = \frac{1}{|C|} < \frac{1}{2m}$.
- Nature also knows $g(\mathbf{x})$, so as a labeling function f, Nature plays a sinister move and chooses $f(\mathbf{x}) = -g(\mathbf{x})$ for all $\mathbf{x} \in \mathcal{X}$, namely, just the opposite of what h_s will predict on unseen points.
- Now, let S be a training sample. Let $supp(S) \subset C$ denote the set of different points $\mathbf{x} \in \mathcal{X}$, that

appears in S. Recall that the same $\mathbf{x}$ may appear in S several times and therefore the $|supp(S)|$ can take any value between 1 and m. Since $|supp(S)| \leq m$ and since $\mathcal{D}$ is uniform over C, we have $\mathcal{D}(\{\mathbf{x} \in \mathcal{X} \setminus supp(S)\}) \geq 1/2$. In other words, the probability that a new test point drawn according to $\mathcal{D}$ was not seen in S is at least $1/2$.

- Now, as the game requires, we feed the sample S into $\mathcal{A}$ and obtain $h_S = \mathcal{A}(S)$. What is the loss? Regardless of what h_S predicts on points seen in S, a test point has probability $\geq 1/2$ to be in the unseen part $\mathcal{X} \setminus supp(S)$. Thus, with probability of at least $1/2$, the rule h_S will predict $h_S(\mathbf{x}) = g(\mathbf{x})$ and will be wrong, since $f(\mathbf{x}) = -g(\mathbf{x})$. Therefore, $L_{\mathcal{D},f}(h_S) \geq 1/2$.
- But this happens for *every* training sample S. So, Nature's strategy ensures that with probability 1 (over the choice of training samples according to $\mathcal{D}$), the game results in a loss $L_{\mathcal{D},f}(h_S) \geq 1/2$.
- So, we can not find a learner $\mathcal{A}$ that will be Probably Approximately correct (for any δ and for $\varepsilon < \frac{1}{2}$) regardless of Nature's strategy $\mathcal{D}, f$. Asking for a larger training sample won't help - if we increase m, Nature will just choose a larger set C and a distribution $\mathcal{D}$ which is uniform over that larger C.

■

What went wrong? Nature could choose *any* labeling function, f, that she wanted, and we tried to learn (to generalize/predict) f from a sample that was too small compared to the number of possible functions Nature could choose from. We find that we just cannot design a Probably Approximately correct learner if the set of possible labeling functions is "too large".

This is known as ***"No Free Lunch" Theorem***[1]: without assuming anything in advance on the label function, f, learning is impossible. Equivalently, if the set of possible labeling function f is too large, then Nature can play a function that we can't learn. Even if we take a larger sample, Nature, in turn, chooses a more complicated f. So if no limitations are placed on Nature's choice of label function, learning is impossible.
Example 4.3 demonstrated the essence of the No Free Lunch Theorem, but was not a proof of it, since we restricted ourselves to $g(\mathbf{x})$ that was independent of S. That restriction made life easier for Nature since it enabled her to choose $f(\mathbf{x}) = -g(\mathbf{x})$ as a label function that maximized the error of our learner. In reality, we could choose different $g_S(\mathbf{x})$'s for each of the different S's that did not contain $\mathbf{x}$, thus limiting Nature's ability to maximize the loss. In any case, example 4.3 gives the crucial bits of intuition which we will need later on.

As mentioned above, there are several theorems that can be called a "No Free Lunch" - basically any theorem that shows that without some prior knowledge on the labeling function, that is, when there are too many possibilities for f, learning it is impossible. Below is an actual, formal, No Free Lunch theorem. Its proof, which uses the notion of Agnostic PAC that we will encounter a bit later, can be found in the book "Understanding Machine Learning" (Theorem 5.1, and exercise 3 in section 5.5 in that book).

Theorem 4.2.1 — No Free Lunch. Let $\mathcal{X}$ be a sample domain, $|\mathcal{X}| = \infty$. For every $0 < \varepsilon < 1/2$, there exists $\delta > 0$ such that, for every algorithm $\mathcal{A}$, there exists a distribution $\mathcal{D}$ over $\mathcal{X}$ and a function $f : \mathcal{X} \to \mathcal{Y}$ for which, when running $\mathcal{A}$ over a sample S of any finite size, drawn *i.i.d* from $\mathcal{D}$, then with probability of at least δ, the output of $\mathcal{A}$, h_S, has a loss larger than ε: $L_{\mathcal{D},f}(h_S) \geq \varepsilon$.

[1] There are in fact several "No Free Lunch" theorems revolving around the same idea

Note that $L_{\mathcal{D},f} = 1/2$ is the loss that one gets from a completely *random guess* of labels. Thus, the condition $\varepsilon < 1/2$ in the above theorem implies that without prior assumptions on f we can not guarantee a prediction which will have a lesser loss than a random guess.

Needing Hypothesis Classes

The No Free Lunch principle implies that to be able to learn, the learner *must* receive enough prior knowledge about the function f. In other words, we should assume that the target f comes from some *hypothesis class*, $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$, or at least that it resembles closely some functions in that class. The class of functions $\mathcal{H}$ should not be too broad or else the problem we encountered in example 4.3 might reappear.

Realizability Assumption

Suppose that a hypothesis class $\mathcal{H}$ is specified for our game, meaning that we restrict the choice of rules that our algorithm can predict to a certain family of functions. The ***realizable case*** is when Nature must play a function $f \in \mathcal{H}$. Actually, we don't care if Nature plays a function f that is not in $\mathcal{H}$ as long as there is a function h^* in $\mathcal{H}$ which is identical to f on all points over which $\mathcal{D}$ does not vanish so that we will never see examples where $f(\mathbf{x}) \neq h(\mathbf{x})$, neither in the training nor in the test samples. So the formal mathematical ***Realizability Assumption*** is this: Nature plays a function f such that there exists $h^* \in \mathcal{H}$ with $L_{\mathcal{D},f}(h^*) = 0$.

The learner is given $\mathcal{H}$ before the learning starts and will only output $h_S \in \mathcal{H}$. In other words, for a training sample of size m the learner (learning algorithm) is a map $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^m \to \mathcal{H}$ such that $\mathcal{A} : S \mapsto h \in \mathcal{H}$. As before, we will continue to abuse notation and write $\mathcal{A}$ instead of $\mathcal{A}_m$, and when we say "the learning algorithm $\mathcal{A}$" we will sometimes mean "a sequence of learners $\{\mathcal{A}_m\}_{m=1}^{\infty}$, one for each possible sample size".

Theorem 4.2.1 told us that if $|\mathcal{X}| = \infty$ then $\mathcal{H} = \mathcal{Y}^{\mathcal{X}}$ is too large to learn. This brings us to ask the following questions:

- What are the "small enough" hypothesis classes $\mathcal{H}$ for which we *can* find a Probably Approximately correct learner? And what are the "too large" hypothesis classes $\mathcal{H}$ for which we *cannot*?
- Assume we have a "small enough" $\mathcal{H}$, in the sense that for every ε, δ we have at least one strategy, $(m, \mathcal{A})$, such that $\mathcal{A}$ is Probably Approximately correct, with accuracy ε and confidence δ, no matter how Nature plays. This means that for every ε, δ there is a *minimal number of training samples*, which we will denote by $m_{\mathcal{H}}(\varepsilon, \delta)$, for which there exists at least one algorithm with the required accuracy and confidence. Can we find this minimal function $m_{\mathcal{H}}(\varepsilon, \delta)$? Is there a relation between the "size" of the hypothesis class $\mathcal{H}$ and $m_{\mathcal{H}}(\varepsilon, \delta)$?
- Assume we have a "small enough" $\mathcal{H}$. Can we find a concrete learner $\mathcal{A}$ that always succeeds in learning functions from $\mathcal{H}$? If yes, how many training samples m does $\mathcal{A}$ need to always succeed in learning a function from $\mathcal{H}$ (always be Probably Approximately correct, no matter how Nature plays)?
- Can we find the *most training-data efficient* learner, namely a learner that can succeed with the minimal number of samples $m_{\mathcal{H}}(\varepsilon, \delta)$ mentioned above?

4.2.1 Learning As A Game - Third Version

To reach a final version of our learning game, we now include the hypothesis class to it.

Definition 4.2.1 ***The Learning Game (third version)***: Fix desired accuracy $\varepsilon \in (0,1)$ and confidence $\delta \in (0,1)$. Fix a hypothesis class $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$.

- We choose a sample size m and a learner $\mathcal{A} : (\mathcal{X},\mathcal{Y})^m \to \mathcal{H}$. Both m and $\mathcal{A}$ can depend on (ε,δ).
- Nature knows our strategy and, after us, chooses strategy that consists of a probability distribution $\mathcal{D}$ over $\mathcal{X}$, and a label function *from the hypothesis class*, $f \in \mathcal{H}$. That is, Nature's strategy depends not only on ε,δ,m and $\mathcal{A}$ but also on the hypothesis class, $\mathcal{H}$. (This rule of the game is somewhat stricter than necessary since, as mentioned above, we could allow Nature to choose a function $f \notin \mathcal{H}$ as long as there exists $h^* \in \mathcal{H}$ with $L_{\mathcal{D},f}(h^*) = 0$).
- A sample S of size m is drawn according to $\mathcal{D}$ and is labeled according to f
- The sample S is fed into $\mathcal{A}$ to produce a prediction rule $h_S = \mathcal{A}(S)$. Note that $h_S \in \mathcal{H}$.
- The payoff is $L_{\mathcal{D},f}(h_S)$. It is random since S is random and therefore h_S is random.
- Nature does her best to win. So we will look for learners $\mathcal{A}$ for which there is a probability of at least $1-\delta$ that the loss $L_{\mathcal{D},f}(h_S)$ will not exceed ε, no matter what strategy $\mathcal{D},f$ Nature might play.
- To determine if we were successful in the game, we play the game many many times (both us and Nature play the same strategies, just the training samples drawn are different). We count and calculate the probability, over the random draws of training samples S, of the event $\{S \sim \mathcal{D}^m \,|\, L_{\mathcal{D},f}(h_S) \leq \varepsilon\}$. If this probability is found to be larger than $1-\delta$, that is, if the learner $\mathcal{A}$ we chose was Probably Approximately correct with accuracy ε and confidence δ - against Nature's best strategy - we say that *we have been successful (with regards to the parameters ε,δ and hypothesis class $\mathcal{H}$).*

4.2.2 Example: Threshold Functions

We saw above that if $|\mathcal{X}| = \infty$ and $\mathcal{H}$ is the class of all functions from $\mathcal{X}$ to $\mathcal{Y}$, namely $\mathcal{H} = \mathcal{Y}^{\mathcal{X}}$, we can't be successful in the third version of the learning game. On the other hand, we know that if we take a very small $\mathcal{H}$, for example, one that contains only a single function, learning is possible and is in fact trivial. So we know that when the hypothesis class is "small enough", it *is* possible to be successful in the third version of the learning game even when the sample space is infinite, which is the reason why learning is possible. What is left to find out is how broad $\mathcal{H}$ can be chosen while maintaining the possibility to learn.

In the following example we will have an infinite (in fact uncountably infinite) sample space, and a very simple hypothesis class. We will see that we can be successful in the third version of the game, for any ε,δ. Consider the domain $\mathcal{X} = \mathbb{R}$, i.e., there is only one feature. We still consider a classification problem but use the label set $\mathcal{Y} = \{0,1\}$ instead of $\{\pm 1\}$, in order to be able to use the standard definition of Threshold Functions.

Definition 4.2.2 — Threshold Functions Hypothesis Class. The set of function:

$$\mathcal{H}_{th} = \{x \mapsto h_\theta(x) : \theta \in \mathbb{R}\}$$

where $h_\theta(x) = \mathbb{1}[x > \theta]$ and $h_\infty(x) = 0, h_{-\infty}(x) = 1$ for all $x \in \mathbb{R}$, is called the Hypothesis Class of ***Threshold Functions*** over $\mathbb{R}$.

For the hypothesis class $\mathcal{H}_{th}$ our learning game takes the following form. Functions in $\mathcal{H}_{th}$ classify

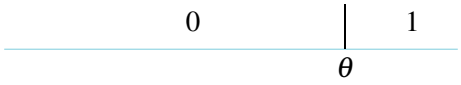


Figure 4.1: *An example of a threshold function*

points on the real line as 0 up to a certain threshold point. Beyond that threshold, they classify all points as 1. Nature chooses one of these functions, that is, it chooses a threshold θ (which is unknown to us) and a distribution $\mathcal{D}$ over the real line. We receive a training sample S of labeled points, and would like to successfully predict the label of future samples. Our job is therefore to determine, as accurately as possible, the unknown cutoff θ.

Now that we were given a hypothesis class, $\mathcal{H}_{th}$, we should specify our strategy, which consists of the number of samples m we need and a learning algorithm $\mathcal{A}$ that will process a training sample and produce a decision rule. As before let $S = \{(x_i, y_i)\}_{i=1}^{m}$ be the training set. As we will see, our choice of learner will not depend on ε or δ but our choice of m will certainly depend on them.

Learning Algorithm for Threshold Functions

The training data may take the form shown in Figure 4.2 below, but not the form shown in Figure 4.3, which is forbidden because it does not obey the Realizability Assumption (the assumption that Nature chooses $h \in \mathcal{H}$).

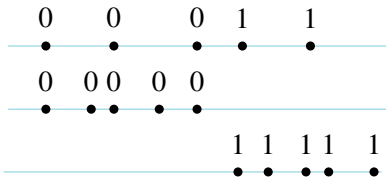


Figure 4.2: *Valid training data for $\mathcal{H}_{th}$*

Figure 4.3: *Invalid training data for $\mathcal{H}_{th}$ - violates the Realizability Assumption*

One possible algorithm, which follows the ERM principle, is the following:

Algorithm 4 Find Threshold Function

Return hypothesis $h_{\theta_{alg}}(x)$ where

- If $y_i = 1$ for all $i = 1, \ldots, m$, then $\theta_{alg} = -\infty$, (the rule classifying all points as 1)
- If $y_i = 0$ for all $i = 1, \ldots, m$, then $\theta_{alg} = +\infty$, (the rule classifying all points as 0)
- In all other cases $\theta_{alg} = \max_i \{x_i : \ y_i = 0\}$

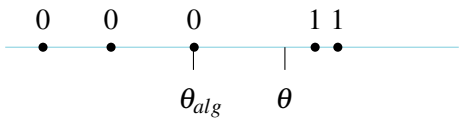


Figure 4.4: *Algorithm for predicting threshold functions*

Number of Samples

Given $\varepsilon, \delta \in (0,1)$ we would like to know what is the sample size, m, that guarantees that, with probability at least $1-\delta$, the true error is at most ε.

Claim 4.2.2 Let $\varepsilon, \delta \in (0,1)$. If $m \geq \frac{\log(1/\delta)}{\varepsilon}$ then for any distribution $\mathcal{D}$ over the real line and any choice of threshold function $f_\theta \in \mathcal{H}_{th}$, with probability of at least $1-\delta$ (over the choice of training sample S of size m), the loss $L_{\mathcal{D},f_\theta}(h_{\theta_{alg}})$ of Algorithm 4 for learning threshold functions is at most ε:

$$\mathcal{D}^m\left\{S \middle| L_{\mathcal{D},f_\theta}\left(h_{\theta_{alg}}\right) \leq \varepsilon\right\} \geq 1-\delta$$

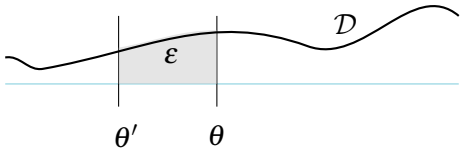
Proof. Let $\mathcal{D}$ be a probability distribution over $\mathbb{R}$. Let $f_\theta \in \mathcal{H}_{th}$ be the true label function $f_\theta \in \mathcal{H}_{th}$. From the properties of the algorithm it follows that:

$$\theta_{alg} \leq \theta$$

The prediction rule produced by the algorithm will be correct for test samples with $x \leq \theta_{alg}$ or with $x > \theta$ and incorrect for $\theta_{alg} < x \leq \theta$. Thus, the true error is given by

$$L_{\mathcal{D},f_\theta}(h_{\theta_{alg}}) = \mathcal{D}(x : \theta_{alg} < x \leq \theta)$$

If $\mathcal{D}(x : -\infty < x \leq \theta) < \varepsilon$ then $\mathcal{D}(x : \theta_{alg} < x \leq \theta) < \varepsilon$ and therefore the true error is *always* (that is, with probability 1, regardless of S or m) smaller than ε which is what we needed to prove. If $\mathcal{D}(x : -\infty < x \leq \theta) \geq \varepsilon$ then there exists θ' such that $\mathcal{D}(x : \theta' < x \leq \theta) = \varepsilon$.



If there is a point $(x_i, y_i) \in S$ with $\theta' < x_i \leq \theta$ then $y_i = 0$ (because $x_i \leq \theta$) and therefore $\theta' \leq \theta_{alg} \leq \theta$. Thus,

$$L_{\mathcal{D},f_\theta}(h_{\theta_{alg}}) = \mathcal{D}(\{x : \theta_{alg} < x \leq \theta\}) \leq \mathcal{D}(\{x : \theta' < x \leq \theta\}) = \varepsilon$$

The probability of *not* getting such sample in the train set is $(1-\varepsilon)^m$ and therefore the probability of having a true error which is larger than ε is not larger than $(1-\varepsilon)^m$. Now, using $m \geq \frac{\log(1/\delta)}{\varepsilon}$ and $1-\varepsilon < e^{-\varepsilon}$ (which holds for $\varepsilon > 0$), we have $(1-\varepsilon)^m < e^{-m\varepsilon} < \delta$. ■

Threshold Functions - Conclusion

We saw that, for $\mathcal{X} = \mathbb{R}, \mathcal{Y} = \{0,1\}$ and $\mathcal{H} = \mathcal{H}_{th}$, we have a strategy (a choice of sample size $m = m_{\mathcal{H}_{th}}(\varepsilon, \delta)$ and a learning algorithm $\mathcal{A}$) that is *always successful against Nature*, for any values ε, δ specified.

Recall that for $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \{0,1\}$ and $\mathcal{H} = \{h |\ h : \mathbb{R} \to \mathbb{R}\}$ there were values of ε, δ for which we *could not be successful* regardless of how we played. In fact, we could not be successful for any $\varepsilon < 1/2$. So whether we can be successful for any ε and δ seems to be a property of the hypothesis class we choose. This leads us to the famous definition of a ***Probably Approximately Correct (PAC)*** learnable hypothesis class.

4.3 PAC Learning

Definition 4.3.1 A hypothesis class, $\mathcal{H}$, is ***PAC Learnable*** if there exists a learning algorithm $\mathcal{A}$ and a function $m_{\mathcal{H},\mathcal{A}} : (0,1)^2 \to \mathbb{N}$ with the following property that:

- For every $\varepsilon, \delta \in (0,1)$
- For every distribution $\mathcal{D}$ over $\mathcal{X}$
- For every labeling function $f : \mathcal{X} \to \{\pm 1\}$ such that there exists $h^* \in \mathcal{H}$ that satisfies $L_{\mathcal{D},f}(h^*) = 0$

when running the learning algorithm $\mathcal{A}$ on $m \geq m_{\mathcal{H},\mathcal{A}}(\varepsilon, \delta)$ *i.i.d* samples generated by $\mathcal{D}$ and labeled by f, the algorithm returns a hypothesis $h_S = \mathcal{A}(S)$ such that, with probability of at least $1 - \delta$ (over the choice of the training samples), we have $L_{\mathcal{D},f}(h_S) \leq \varepsilon$:

$$\mathcal{D}^m \left(\left\{ S \middle| L_{\mathcal{D},f}(h_S) \leq \varepsilon \right\} \right) \geq 1 - \delta$$

Denote the minimal sample size required for the above definition to hold with respect to ε, δ and with respect to any algorithm, by

$$m_{\mathcal{H}}(\varepsilon, \delta) = \min_{\mathcal{A}} m_{\mathcal{H},\mathcal{A}}(\varepsilon, \delta)$$

The function $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ is called the ***Sample Complexity*** of the PAC learnable hypothesis class $\mathcal{H}$.

R Note that PAC learnability is only one possible definition of learning that can account for the fundamental limitations on accuracy and confidence. We could, for example, settle for a specific, "good enough", values of δ and ε, instead of requiring that the above condition holds for *any* $\varepsilon, \delta \in (0,1)$. Such *weak learners* will be discussed in later chapters.

4.3.1 PAC Learnability of Finite Hypothesis Classes

To better understand when a hypothesis class is PAC learnable, let us first consider the case where $\mathcal{H}$ is a *finite* hypothesis class. Though finite, this hypothesis class is still very large. For example, consider $\mathcal{H}$ as the set all the functions from $\mathcal{X}$ to $\mathcal{Y}$ that can be implemented using a Python program of length at most b, for b fixed and large. Or, take $\mathcal{H}$ to be all the functions from $\mathcal{X}$ to $\mathcal{Y}$ where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite.

One might expect that there would not be much to say in this generality, without specific details of $\mathcal{X}$ and $\mathcal{H}$. Surprisingly, it turns out that there is a simple type of learner, called **Empirical Risk Minimization**, that is always successful on finite hypothesis classes and in fact on many other hypothesis classes that we will encounter later. The idea behind these powerful learners is very natural: try to be as correct as possible on the training data. In other words, find the function h in $\mathcal{H}$ that gives the correct label to the maximal number of points in the training sample, S.

Definition 4.3.2 For $h \in \mathcal{H}$ and $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ we define the ***empirical risk*** by

$$L_S(h) = \frac{1}{m}|\{i : h(\mathbf{x}_i) \neq y_i\}|.$$

A rule, h, with $y_i = h(\mathbf{x}_i)$ for $i = 1, \ldots, m$, that is, with $L_S(h) = 0$, is called ***Consistent*** with the training sample, S.

Definition 4.3.3 An algorithm that, for each S, predicts a rule that minimizes the empirical risk $L_S(h)$, is called an ***Empirical Risk Minimization (ERM)*** learner and is denoted by $ERM_{\mathcal{H}}$. That is,

$$ERM_{\mathcal{H}} : S \mapsto \operatorname*{argmin}_{h \in \mathcal{H}} L_S(h)$$

Notice that since $L_S(h) \geq 0$ and we are minimizing over a finite class, a minimum exists. For each given S, the minimum may be obtained by more than one h in $\mathcal{H}$, in which case $ERM_{\mathcal{H}}$ actually refers to a *set of algorithms*, each of which returns one of the minimizers. *Under the realizability assumption* we know that the lower bound $L_S(f) = 0$ is achievable. In other words, under our assumption that $f \in \mathcal{H}$, an ERM learner will always return a consistent rule. Hence, in the realizable case, the terms "consistent rule" or "ERM rule" are synonyms.

Learning Finite Classes

Theorem 4.3.1 Let $\varepsilon, \delta \in (0,1)$, $|\mathcal{H}| < \infty$, $m \geq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$ and let h_S^{ERM} be the prediction rule of an $ERM_{\mathcal{H}}$ learner. Then for every $f \in \mathcal{H}$ and every $\mathcal{D}$, with probability of at least $1-\delta$ (over the choice of S of size m), $L_{\mathcal{D},f}(h_S^{ERM}) \leq \varepsilon$.

Proof. Let $\varepsilon \in (0,1)$, $\mathcal{D}$ a probability distribution over $\mathcal{X}$ and a labeling function f. By specifying $\varepsilon, \mathcal{D}$ and f we implicitly define define a subset of $\mathcal{H}$, which we will denote as $\mathcal{H}_B$ ("B" for "bad"), containing all the "bad" h's. That is, all hypothesis that do not approximate f well:

$$\mathcal{H}_B = \left\{ h \in \mathcal{H} \,\middle|\, L_{\mathcal{D},f}(h) > \varepsilon \right\}$$

By definition

$$\left\{ S \middle| L_{\mathcal{D},f}\left(h_S^{ERM}\right) > \varepsilon \right\} = \left\{ S \middle| h_S^{ERM} \in \mathcal{H}_B \right\}$$

and therefore, to prove theorem 4.3.1, we have to show that

$$\mathcal{D}^m\left(\left\{ S \middle| h_S^{ERM} \in \mathcal{H}_B \right\}\right) < \delta$$

.

Any sample S defines a subset of $\mathcal{H}$, which we will denote as $\mathcal{H}_C^S$ ("C" for "consistent"), containing all h's that are consistent with f on S:

$$\mathcal{H}_c^S = \{h:\ L_S(h) = 0\} = \{h:\ h(\mathbf{x}_i) = f(\mathbf{x}_i),\ i = 1\ldots,m\}$$

Any sample S, also defines an intersection set $\mathcal{H}_{BC}^S = \mathcal{H}_B \cap \mathcal{H}_C^S$ which contains all h's that are consistent *and* bad (Figure 4.5).
Our algorithm is an ERM learner, $h_S^{ERM} \in \mathcal{H}_C^S$. Therefore all the samples S, in the set $\left\{S : h_S^{ERM} \in \mathcal{H}_B\right\}$ are such that the intersection set, $\mathcal{H}_{BC}^S$, is not empty since it contains at least one function, namely, h_S^{ERM}. Thus,

$$\left\{S \middle| h_S^{ERM} \in \mathcal{H}_B\right\} \subseteq \left\{S \middle| \mathcal{H}_{BC}^S \neq \varnothing\right\}$$

As such it is sufficient to prove that:

$$\mathcal{D}^m\left(\left\{S : \mathcal{H}_{BC}^S \neq \varnothing\right\}\right) < \delta$$

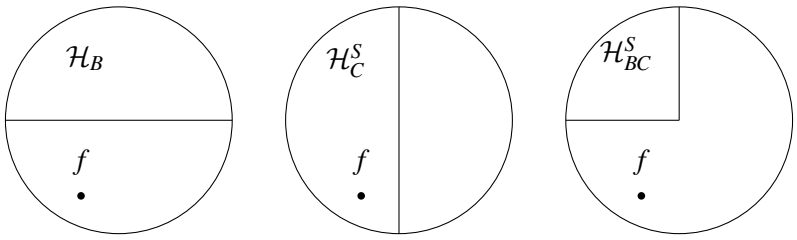


Figure 4.5: *The Bad ($\mathcal{H}_B$) and S-Consistent ($\mathcal{H}_C^S$) subsets of $\mathcal{H}$ and their intersection set ($\mathcal{H}_{BC}^S$)*

Note that

$$\left\{S \middle| \mathcal{H}_{BC}^S \neq \varnothing\right\} = \bigcup_{h\in\mathcal{H}} \left\{S \middle| h \in \mathcal{H}_{BC}^S\right\} = \bigcup_{h\in\mathcal{H}_B} \left\{S \middle| h \in \mathcal{H}_C^S\right\}$$

The first equality is simply a way of saying that instead of checking each S to see if its related intersection set, $\mathcal{H}_{BC}^S$, is non-empty, and then adding it to the above set, we can check each h in $\mathcal{H}$ to see if it appears in any of the $\mathcal{H}_{BC}^S$'s, and each time it does, we add the S that defined that particular $\mathcal{H}_{BC}^S$ to the set. The second equality results from the definition $\mathcal{H}_{BC}^S = \mathcal{H}_B \cap \mathcal{H}_C^S$. We are therefore left with proving that:

$$\mathcal{D}^m\left(\bigcup_{h\in\mathcal{H}_B} \left\{S \middle| h \in \mathcal{H}_C^S\right\}\right) < \delta$$

Using the union bound we get that

$$\mathcal{D}^m\left(\bigcup_{h\in\mathcal{H}_B} \left\{S \middle| h \in \mathcal{H}_C^S\right\}\right) \leq \sum_{h\in\mathcal{H}_B} \mathcal{D}^m\left(\left\{S \middle| h \in \mathcal{H}_C^S\right\}\right)$$

Given a hypothesis h, $\mathcal{D}^m\left(\left\{S|h \in \mathcal{H}_C^S\right\}\right)$ is the probability to pull a sample over which h is perfectly correct (has the right labels for all $\mathbf{x}_i$'s). Since the samples are drawn independently, this probability equals the probability that h will be correct for each $\mathbf{x}_i$ separately. The probability that h will be

incorrect for a random $\mathbf{x}$ is exactly $L_{\mathcal{D},f}(h)$ and therefore the probability to be correct for m such $\mathbf{x}$'s is $\left(1-L_{\mathcal{D},f}(h)\right)^m$. We therefore have, for any h,

$$\mathcal{D}^m\left(\left\{S\middle|h\in\mathcal{H}_C^S\right\}\right)=\left(1-L_{\mathcal{D},f}(h)\right)^m$$

and in particular, if $h\in\mathcal{H}_B$ then $L_{\mathcal{D},f}(h)>\varepsilon$ which means that

$$\mathcal{D}^m\left(\left\{S\middle|L_S(h)=0\right\}\right)<(1-\varepsilon)^m\quad\forall h\in\mathcal{H}_B$$

Thus we obtain

$$\mathcal{D}^m\left(\bigcup_{h\in\mathcal{H}_B}\left\{S\middle|h\in\mathcal{H}_C^S\right\}\right)\leq\sum_{h\in\mathcal{H}_B}(1-\varepsilon)^m<|\mathcal{H}_B|\cdot(1-\varepsilon)^m\leq|\mathcal{H}|\cdot(1-\varepsilon)^m$$

Finally, using $1-\varepsilon\leq e^{-\varepsilon}$ we conclude:

$$\mathcal{D}^m\left(\left\{S\middle|L_{\mathcal{D},f}(ERM_{\mathcal{H}}(S))>\varepsilon\right\}\right)<|\mathcal{H}|\,e^{-\varepsilon\cdot m}$$

If specifying $m\geq\frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$, the right-hand side would be smaller δ which concludes the proof. ■

Theorem 4.3.1 means that, by definitions 4.3.1 and 4.1.6 any *finite* hypothesis class $\mathcal{H}$ is PAC-learnable with a sample complexity not larger than $\log(|\mathcal{H}|/\delta)/\varepsilon$. Moreover, for any m greater or equal to that lower bound, any $ERM_{\mathcal{H}}$ learner is a PAC learner.

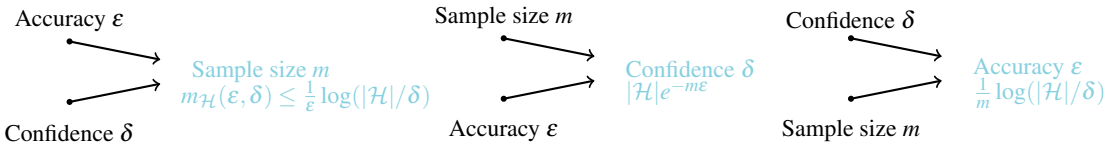


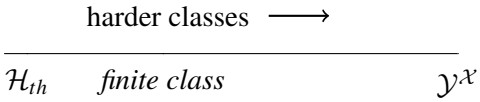
Figure 4.6: *Relation between m, ε and δ*

We have therefore shown that any finite hypothesis class $\mathcal{H}$ is PAC learnable using any ERM learning algorithm, and has a sample complexity $m_{\mathcal{H}}(\varepsilon,\delta)\leq\log(|\mathcal{H}|/\delta)/\varepsilon$. In practical terms, this means that whenever we are given $\varepsilon,\delta\in(0,1)$, a finite hypothesis class $\mathcal{H}$, and a sample of size of at least $\log(|\mathcal{H}|/\delta)/\varepsilon$, we can simply scan $\mathcal{H}$ to find a hypothesis that labels the points in S correctly. With probability of at least $1-\delta$ the generalization error of that hypothesis will not exceed ε.

Several natural questions may now come to mind:

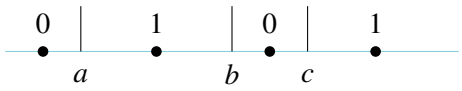
- Is the bound $m_{\mathcal{H}}(\varepsilon,\delta)\leq\frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$ tight? In other words, can the ERM learner, or an other learner, be Probably Approximately correct (with accuracy ε and confidence δ) using fewer than $\frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$ samples?
- What happens when noise is present so the y's are not deterministically determined by x?
- What happens when our hypothesis class is infinite?

Consider the last question. As we have seen, the class of threshold functions over $\mathbb{R}$, $\mathcal{H}_{th}$, in spite of being infinite, *is* PAC learnable, with sample complexity $m_{\mathcal{H}_{th}}(\varepsilon,\delta) \leq \frac{\log(1/\delta)}{\varepsilon}$, which is obtained by using an $ERM_{\mathcal{H}_{th}}$ learning rule (recall that the rule output by our algorithm was consistent on any sample). So $\mathcal{H}_{th}$ appears to be simple to learn, even simpler, in terms of the sample complexity, than a finite class. While $\mathcal{Y}^{\mathcal{X}}$ is too complex to learn. Can we explain why? What we need in order to answer the above questions systematically is some sort of a *complexity measure* with which we can order classes by their difficulty along the complexity axis shown in the figure below.



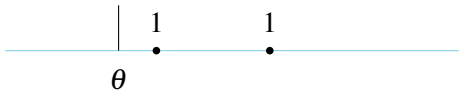
For example, consider the ***Two-Intervals*** hypothesis class, defined by

$$\mathcal{X} = \mathbb{R}, \quad \mathcal{H} = \{h_{a,b,c} : a < b < c \in \mathbb{R}\}, \quad h_{a,b,c} = \mathbb{1}[x \in [a,b] \vee x \geq c]$$

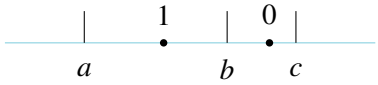


Suppose we would like to learn with the threshold function class, $\mathcal{H}_{th}$, the following sample:

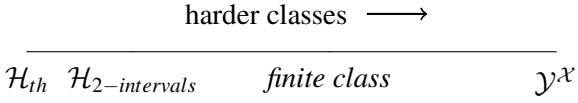
A possible answer would be:



However, samples where $x_1 < x_2$ and $y_1 = 1, y_2 = 0,$, as in the figures below, can not be learned (consistently) by $\mathcal{H}_{th}$, although it can be learned with the 2-intervals class:



Indeed, we somehow feel that the 2-Interval class has a larger complexity than that of $\mathcal{H}_{th}$ but smaller than that of finite classes.



4.3.2 VC Dimension

The VC-Dimension is a*measure of complexity* of hypothesis classes. It is called a *combinatorial* measure because it relies on a certain way of counting the possibilities available in the hypothesis class to label points in $\mathcal{X}$. This measure provides a *precise* test for whether or not a hypothesis class is simple enough to learn in the sense of PAC learnability. It also enables the calculation of clear bounds on the sample complexity of a simple hypothesis class $\mathcal{H}$.

Suppose we receive a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ and were able to fully explain the labels using a hypothesis from a class $\mathcal{H}$, namely, to find a function $h \in \mathcal{H}$ with empirical risk $L_S(h) = 0$. Suppose now, only to see what will happen, we deliberately corrupt our sample S by changing 'by hand' certain labels, and denote the corrupted sample by S'.
Suppose that we *also* succeed in explaining S' using a different hypothesis from the same class $\mathcal{H}$, namely, find another function $h' \in \mathcal{H}$ with $L_{S'}(h') = 0$. If we are able to do that, no matter what labels we choose to corrupt and regardless of the sample size, it means that something isn't right. How can we hope to generalize based on a training sample S if, regardless of the labels in S, we can find $h \in \mathcal{H}$ with $L_S(h) = 0$?

Definition 4.3.4 — Restriction. Let $C \subseteq \mathcal{X}$ be a subset of the sample space, $\mathcal{X}$, and let $h \in \mathcal{H}$ be a hypothesis. The function $h_C : C \to \mathcal{Y}$, defined as: $\forall \mathbf{x} \in C$, $h_C(\mathbf{x}) = h(\mathbf{x})$, is called the ***restriction*** of h to C. The set $\mathcal{H}_C = \{h_C \,|\, h \in \mathcal{H}\}$ of the restrictions of the h's in $\mathcal{H}$ is called the restriction of $\mathcal{H}$ to C.

R Since $\mathcal{Y} = \{\pm 1\}$, we can represent each h_C by the vector $(h(x_1), \ldots, h(x_{|C|})) \in \{\pm 1\}^{|C|}$. The number of possible such vectors is $2^{|C|}$, therefore $|\mathcal{H}_C| \leq 2^{|C|}$.

To further clarify the above point, consider the following important observation. Suppose that $\mathcal{H}$ contains *all* the possible functions over a set $C \subset \mathcal{X}$ of size m, that is, $\mathcal{H}_C = \mathcal{Y}^C$, then there is no Probably Approximately Correct learner that uses $m/2$ or fewer training samples. To understand why, recall example 4.3 that demonstrated the argument behind the No Free Lunch theorem. To play our game against Nature, we choose a learner $\mathcal{A}$ and ask for a training sample size of $m/2$ or less. For points $\mathbf{x} \in \mathcal{X}$ not seen in the training set, we choose to predict $g(\mathbf{x})$, where $g : \mathcal{X} \to \mathcal{Y}$. We only have to make sure that $g(\mathbf{x})$ is such that the resulting h_S will belong to $\mathcal{H}$.

As in example 4.3, Nature plays a distribution $\mathcal{D}$ that is uniform over C and zero elsewhere, and a labeling function f such that $f_C(\mathbf{x}) = -g_C(\mathbf{x})$, that is, $f(\mathbf{x}) = -g(\mathbf{x})$ for any $\mathbf{x} \in C$ (Since $\mathcal{D}$ vanishes outside of C, Nature doesn't really care how her f is defined outside C). Nature can *always* choose such f, since $\mathcal{H}_C = \mathcal{Y}^C$ so in particular $-g_C(\mathbf{x}) \in \mathcal{H}_C$, which means that there is at least one $h \in \mathcal{H}$ with $h_C(\mathbf{x}) = -g_C(\mathbf{x})$. Again, as in example 4.3, our learner fails since the loss will be $1/2$ or more - regardless of the training sample (as long as it is of size $m/2$ or smaller).
As in example 4.3, the argument presented here is not a full proof since we restricted our choice of algorithm only to such that chooses the *same* label $g(\mathbf{x})$, for *all* S's in which $\mathbf{x}$ does not appear. However, the intuition that our argument implies is correct: As long as $\mathcal{H}$ contains *any* set C of size $2m$ with the property that $\mathcal{H}_C = \mathcal{Y}^C$, then we cannot learn with a training sample of size m. It follows that the *maximal size* of such a set C in $\mathcal{H}$ is a *critical quantity*: (i) it gives us a lower bound

on $m_{\mathcal{H}}$, the minimal sample size needed, and (ii) if the maximal size is ∞, namely, if for any $m \in \mathbb{N}$, $\mathcal{X}$ contains such as set C with $|C| > m$, $\mathcal{H}$ is not PAC-learnable.

Definition 4.3.5 — Shattering. Let $C = \{\mathbf{x}_1, \ldots, \mathbf{x}_{|C|}\} \subset \mathcal{X}$, $\mathcal{Y} = \{\pm 1\}$ and $\mathcal{H}_C$ be the restriction of $\mathcal{H}$ to C. We say that $\mathcal{H}$ ***shatters*** C if $|\mathcal{H}_C| = 2^{|C|}$, which is equivalent to saying that $\mathcal{H}_C = \mathcal{Y}^C$.

Definition 4.3.6 — VC-dimension. The ***VC-dimension*** of the hypothesis class $\mathcal{H}$ is defined as

$$VCdim(\mathcal{H}) = \max\{|C| \ : \ \mathcal{H} \ \text{ shatters } \ C\}$$

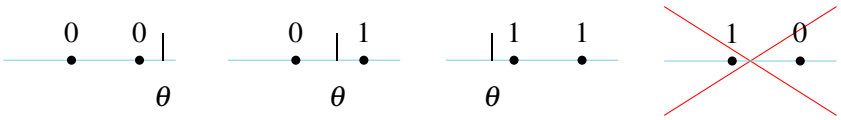
that is, the VC dimension is the maximal size of a set $C \subset \mathcal{X}$ such that $\mathcal{H}_C = \mathcal{Y}^C$.

According to the above definition, in order to show that $VCdim(\mathcal{H}) = d$ we need to show that:

1. There *exists* a set C of size d which is shattered by $\mathcal{H}$.
2. for *any* set C of size greater than d, C is not shattered by $\mathcal{H}$.

■ **Example 4.4 — Threshold class.** Consider the hypothesis class of thereshold function $\mathcal{H}_{th}$. The set $\{0\}$ is shattered by $\mathcal{H}_{th}$ since $x = 0$ can receive both the label 0 and 1, depending on the location of the threshold θ.

In contrast to the above, no two points, x_1 and x_2 can be shattered because if $x_1 < x_2$ then $y_1 \leq y_2$ and therefore the pair of labels $y_1 = 1, y_2 = 0$ is forbidden.



■

■ **Example 4.5 — One-Interval hypothesis class.** Consider the *One-Interval hypothesis class* over $\mathcal{X} = \mathbb{R}$ defined as

$$\mathcal{H} = \{h_{a,b} : a < b \in \mathbb{R}\}, \quad h_{a,b}(x) = \mathbb{1}[x \in [a,b]]$$

Now, take for example the two points $\{0,1\}$. We can place the interval over both, over any one of them, or outside $[0,1]$. Therefore, $\{0,1\}$ is shattered. However, any three points cannot be shattered. Let $x_1 < x_2 < x_3$, then no single interval can cover x_1 and x_3 without containing also x_2 and therefore the labeling $y_1 = 1, y_2 = 0, y_3 = 1$ is forbidden. ■

4.3.3 The Fundamental Theorem of Statistical Learning

In the previous sections, we worked hard to understand two fundamental definitions: PAC-Learnability (and sample complexity) of a hypothesis class, and VC-dimension of a hypothesis class.

Along the way, we saw some connections between these two definitions:

- We saw that a finite hypothesis class is PAC-learnable (using the ERM learners) with sample complexity
$$m_{\mathcal{H}}(\varepsilon,\delta) \leq \frac{\log(|\mathcal{H}|)+\log(1/\delta)}{\varepsilon}$$
and also that in this case $VCdim(\mathcal{H}) \leq \log_2(|\mathcal{H}|)$. These results indicate that there might be a general relation, valid for other hypothesis classes as well, between an *upper bound* on $m_{\mathcal{H}}$ and $VCdim(\mathcal{H})$.
- We saw, using the same argument that led us to the No Free Lunch theorem, that $VCdim(\mathcal{H})$ also gives a *lower bound* on the sample complexity $m_{\mathcal{H}}$ of a hypothesis class $\mathcal{H}$, and that if $VCdim(\mathcal{H})$ is infinite, that class is not PAC-Learnable.

The surprising, wonderful, truth is that VC-dimension gives a complete characterization of PAC-learnability and sample complexity of a hypothesis class, and gives a decisive answer to all the questions we posed at various stages along the way (such as which classes are PAC-learnable, with what sample complexity, and with what algorithm, and is there an algorithm that uses the minimal possible sample size.) These facts result from the ***The Fundamental Theorem of Statistical Learning*** which states the following:

- The PAC-learnability of a hypothesis class is characterized by its *VC dimension*, a combinatorial property that denotes the maximal size of a sample that can be shattered by the class.
- A hypothesis class is PAC-learnable *if and only if* its VC-dimension is finite.
- When $VCdim(\mathcal{H})$ is finite, $\mathcal{H}$ has a finite sample complexity which is, up to multiplicative constants, given by
$$m_H(\varepsilon,\delta) \sim \frac{VCdim(\mathcal{H})+\log(1/\delta)}{\varepsilon}$$
- The ERM learning rule is a generic (near) optimal learner, in the sense that when a hypothesis class is PAC-learnable, an ERM learner using
$$m(\varepsilon,\delta) \sim \frac{VCdim(\mathcal{H})\log(1/\varepsilon)+\log(1/\delta)}{\varepsilon}$$
is a Probably Approximately correct learner with accuracy ε and confidence δ.

Theorem 4.3.2 — The Fundamental Theorem of Statistical Learning. Let $\mathcal{H}$ be a hypothesis class of binary classifiers with VC-dimension $d \leq \infty$. Then, $\mathcal{H}$ is PAC-learnable if and only if $d < \infty$. In this case: there are absolute constants C_1, C_2 (that is, they are independent of d, ε and δ) such that the sample complexity of $\mathcal{H}$ satisfies
$$C_1\frac{d+\log(1/\delta)}{\varepsilon} \leq m_{\mathcal{H}}(\varepsilon,\delta) \leq C_2\frac{d\log(1/\varepsilon)+\log(1/\delta)}{\varepsilon}$$
Furthermore, the upper bound on sample complexity is achieved by the ERM learner.

As we saw, the intuition behind the lower bound was based on how the VC-dimension $VCdim(\mathcal{H})$ was related to the *minimal* number of training samples needed to PAC-learn the hypothesis class $\mathcal{H}$.

To understand the upper bound, we need to understand how is it that the ERM learner is a *generic learning algorithm* that is able to PAC-learn $\mathcal{H}$ with a training sample size related to the VC-dimension $VCdim(\mathcal{H})$.

Before discussing the upper bound, we shall extend our theoretic framework to make it much more flexible and realistic.

4.4 Agnostic PAC

The theoretical framework we developed so far has several serious limitations when it comes to real-world learning problems.

- *It doesn't model noisy labels*: We have no model for measuring errors in labels. In practice, sometimes even though the label for some $x \in \mathcal{X}$ should have been for example 1, it can be measured as -1 due to measurement mistake, noise, etc. We want the learning framework to allow for the fact that we may, with low probability, observe the point $x \in \mathcal{X}$ twice, and get two different labels.
- *The realizability assumption is unrealistic*: The hypothesis class is, after all, only an intelligent guess, and it is unrealistic to assume that the true label function will belong to it.
- *Limited to misclassification loss*: In the previous sections we measured the performance of a classifier using the misclassification loss (the 0-1 loss). We would like to be able to measure performance using any loss function.

To address these limitations we introduce the ***Agnostic PAC*** framework. In addition to address these limitations we can also prove that the the fundamental theorem of statistical learning holds in the Agnostic PAC framework as well.

4.4.1 Introducing the Joint Probability Distribution Over $\mathcal{X} \times \mathcal{Y}$

In the PAC framework, $\mathcal{D}$ is a probability distribution over the sample space $\mathcal{X}$ and the labels are determined deterministically using the label function f. In the new, more general framework, $\mathcal{D}$ will be a probability distribution over $\mathcal{X} \times \mathcal{Y}$.

This means that when we draw a new random example $(\mathbf{x}, y)$ - whether for the training sample S or as a test sample - there is randomness in *both* $\mathbf{x}$ and y which are now *dependent* random quantities.

We can factor $\mathcal{D}$ in two ways, conditioning on $\mathbf{x}$ or on y, both are useful for our understanding. Let (X,Y) be a random variable taking valued in $\mathcal{X} \times \mathcal{Y}$ whose distribution is $\mathcal{D}$.

- $\mathbb{P}(X = \mathbf{x}, Y = y) = \mathbb{P}(X = \mathbf{x})\mathbb{P}(Y = y|X = \mathbf{x})$. From this perspective, this is a direct generalization of our previous framework, where $\mathbf{x}$ was random and $y = f(\mathbf{x})$. Indeed, we draw $\mathbf{x}$ from the marginal distribution with probability $\mathbb{P}(X = \mathbf{x})$ - as we did in the previous framework. We then choose a corresponding label according to the conditional probability $\mathbb{P}(Y = y|X = \mathbf{x})$. Since the marginal random variable Y describing the label is a Bernoulli random variable, one can think of it as a result of a coin flip of a biased coin, with a probability $p(\mathbf{x})$ to obtain $+1$, where the function $p : \mathcal{X} \to [0,1]$ is defined by $\mathbb{P}(Y = +1|X = \mathbf{x}) = p(\mathbf{x})$. If $p(\mathbf{x}) = 0$ or $p(\mathbf{x}) = 1$ for some $\mathbf{x}$, the label is deterministic and we are back to the label function f. But for other values of $p(\mathbf{x})$, whereas before the label depended deterministically on $\mathbf{x}$, now it is random. This models *measurement noise* - the fact that there may be noise in the labels and

that the distribution of the noise may change from one $\mathbf{x}$ to another. This attitude is similar to that of the Logistic Regression classifier, although in that case we did not pay attention to the distribution on $\mathbf{x}$ - instead, we assumed the samples are given and tried to estimate the conditional probability $\mathbb{P}(Y=+1|X=\mathbf{x})=p(\mathbf{x})$.

- $\mathbb{P}(X=\mathbf{x},Y=y)=\mathbb{P}(Y=y)\mathbb{P}(X=\mathbf{x}|Y=y)$. From this perspective, we first draw the label according to a "coin flip" - a Bernoulli random variable. Then, each label has its own distribution for the samples $\mathbf{x}$. We draw the sample $\mathbf{x}$ from $\mathbb{P}(X=\mathbf{x}|Y=+1)$ or from $\mathbb{P}(X=\mathbf{x}|Y=-1)$, according to the label y we obtained. This is a similar approach to the one used in the LDA classifier.

When $\mathcal{D}$ was a distribution over $\mathcal{X}$ alone, we defined the misclassification loss, (4.1), as the probability of obtaining an $\mathbf{x}$ whose *correct* label, $f(\mathbf{x})$, differs from the predicted one, $h(x)$. Now, with $\mathcal{D}$ as a distribution over $\mathcal{X}\times\mathcal{Y}$, we generalize simply by defining the loss as the probability of obtaining an $\mathbf{x}$ whose *measured* label, y, will differ from the predicted one, $h(\mathbf{x})$:

$$L_{\mathcal{D}}(h)=\mathbb{P}_{(\mathbf{x},y)\sim\mathcal{D}}\{h(\mathbf{x})\neq y\}=\mathcal{D}\{(\mathbf{x},y)\,|\,h(\mathbf{x})\neq y\} \tag{4.2}$$

4.4.2 Relaxing Realizability Assumption

Recall that in the case of deterministic labeling, under the realizability assumption, we could reach zero generalization error:

$$\min_{h\in\mathcal{H}} L_{\mathcal{D},f}(h)=L_{\mathcal{D},f}(f)=0$$

However, according to (4.2), in order to calculate the loss, we now have to compare a deterministic values, $h(\mathbf{x})$, to a random one, the measured y, and therefore we can no longer expect zero loss. In other words, we no longer have a “ground truth” labeling function f, at least not one accessible by measurement. The closest thing we have to f is the conditional probability $\mathbb{P}(Y=y|X=\mathbf{x})$).
So the realizability assumption is no longer practical in the sense that we no longer expect to be able to reach 0 generalization loss. Even if a certain underlying true-label function does exist, and no matter how close our predicted rule, $h(\mathbf{x})$, resembles it, we may end up with a non-negligible loss due to the noise. This means we have to change also the definition of *accuracy*: we would like the learning algorithm to output a rule which has generalization loss at most ε *above the minimal possible loss* $\min_{h\in\mathcal{H}} L_{\mathcal{D}}(h)$. We now proceed to identify such a lower bound for the loss.

Definition 4.4.1 Let $\mathcal{D}$ be a probability distribution over $\mathcal{X}\times\mathcal{Y}$. We define the ***Bayes Optimal Classifier*** by

$$f_{\mathcal{D}}(\mathbf{x})=\begin{cases}1 & \mathbb{P}(y=1|\mathbf{x})\geq 1/2\\ 0 & otherwise\end{cases}$$

Note that although $f_{\mathcal{D}}:\mathcal{X}\to\mathcal{Y}$ is a hypothesis, it depends on $\mathcal{D}$, which according to the rules of the game, we don’t know. So $f_{\mathcal{D}}$ is what is known as an ***Oracle Quantity***: if we had an oracle telling us $\mathcal{D}$, then we could classify with $f_{\mathcal{D}}$. Oracle quantities, like this one, are used to compare the loss of any other rule to the loss of the *best possible* rule.

Definition 4.4.2 Let $\varepsilon>0$. We say that a rule $h\in\mathcal{H}$ is ***Approximately Correct*** with ***accuracy*** ε with respect to the distribution $\mathcal{D}$ on $\mathcal{X}\times\mathcal{Y}$ if $L_{\mathcal{D}}(h)$ is as most ε away from the best possible

loss achievable by *any* hypothesis in $\mathcal{H}$:

$$L_{\mathcal{D}}(h) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \varepsilon$$

Notice once more that in our previous framework, under the realizability assumption, the minimal loss was simply 0 since we *assumed* the existence of some $h' \in \mathcal{H}$ with $L_{\mathcal{D}}(h') = 0$. Thus, we let go of the realizability assumption: we no longer assume that there exists a "correct" labeling function and in particular no longer assume that Nature plays a labeling function in the chosen hypothesis class $\mathcal{H}$. Nature's strategy consists only of choosing the joint distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

4.4.3 General Loss Function

The last extension of the framework developed in the previous sections is to allow the use of a general loss function.

Definition 4.4.3 A ***Loss Function*** is a function $\ell : \mathcal{H} \times Z \to [0, \infty)$, where $Z = \mathcal{X} \times \mathcal{Y}$.

R Instead of writing $\ell(h, z)$, we shall often write $\ell(h, (\mathbf{x}, y))$ or $\ell(h(\mathbf{x}), y)$.

We already used the most common example of a classification loss function - the misclassification loss, also known as the *0-1 loss*:

$$\ell_{0,1}(h, (\mathbf{x}, y)) := \begin{cases} 1 & h(\mathbf{x}) \neq y \\ 0 & h(\mathbf{x}) = y \end{cases}$$

The definition, (4.2), of the generalization loss we have been using, $L_{\mathcal{D}}(h)$, can be rewritten in terms of $\ell_{0,1}$ as:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{\mathcal{D}}[\ell_{0,1}(h, (\mathbf{x}, y))]$$

where $\mathbb{E}_{\mathcal{D}}[\cdot]$ denotes the expected value according to $(\mathbf{x}, y) \sim \mathcal{D}$.
Written in its new form, the above definition can be naturally extended to include any loss function:

Definition 4.4.4 — Generalization Loss. Given a distribution $\mathcal{D}$ over $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ and a general loss function $\ell : \mathcal{H} \times \mathcal{Z} \to [0, \infty)$, we define the Generalization Loss, $L_{\mathcal{D}}(h)$, of a Hypothesis h induced by ℓ with respect to $\mathcal{D}$, as the expected value (according to $z \sim \mathcal{D}$) of ℓ:

$$L_{\mathcal{D}}(h) = \mathbb{E}_{\mathcal{D}}[\ell(h, z)]$$

Since definition 4.4.2 did not refer explicitly to the choice of a loss function, we shall keep it as our definition of an approximately correct learner and of the accuracy ε, also in the case of a general (not necessarily 0-1) loss function.

4.4.4 Agnostic-PAC Learnability

The new, more general framework we develop is called ***Agnostic-PAC***.

Definition 4.4.5 Let $\varepsilon, \delta \in (0,1)$. We say that a learner, $\mathcal{A}$, is ***Agnostic Probably Approximately Correct*** (Agnostic-PAC) with a confidence δ and an accuracy ε, ***with respect to*** a loss function ℓ, hypothesis class $\mathcal{H}$ and a distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$, if the probability of obtaining a training sample, S, for which $\mathcal{A}$ will output a prediction rule, $h_S \in \mathcal{H}$, with a loss, $L_{\mathcal{D}}(h_S)$, of at most ε away from the best possible loss achievable by *any* hypothesis in $\mathcal{H}$, is larger or equal to $1-\delta$:

$$\mathcal{D}^m\left(\left\{S \middle| L_{\mathcal{D}}(h_S) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \varepsilon\right\}\right) \geq 1-\delta$$

Definition 4.4.6 — Agnostic PAC Learnability. A hypothesis class $\mathcal{H}$ is Agnostic-PAC learnable with respect to loss $\ell : \mathcal{H} \times (\mathcal{X} \times \mathcal{Y}) \to [0,\infty)$ if there exists a function $\tilde{m}_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^m \to \mathcal{H}$ with the following property:

- For any $\varepsilon, \delta \in (0,1)$
- For any distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$

when running the learning algorithm $\mathcal{A}$ on $m \geq \tilde{m}_{\mathcal{H}}(\varepsilon, \delta)$ *i.i.d* samples degenerated by $\mathcal{D}$, the algorithm returns a hypothesis $h_S = \mathcal{A}(S)$ such that, with probability of at least $1-\delta$:

$$\mathcal{D}^m\left(\left\{S \middle| L_{\mathcal{D}}(h_S) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon\right\}\right) \geq 1-\delta$$

It can be shown that an Agnostic-PAC learner with $\mathcal{A}$ is a PAC learner (see Ex.5). Using Agnostic-PAC learnability , let us write the "learning game" in the Agnostic PAC framework. Let $\varepsilon, \delta \in (0,1)$ be the desired accuracy and confidence levels, let $\mathcal{H}$ be a hypothesis class $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ and a loss function ℓ. We play a game against Nature, with random payoff.

- We choose a sample size m and a learner $\mathcal{A} : (\mathcal{X}, \mathcal{Y})^m \to \mathcal{H}$ where m and $\mathcal{A}$ can depend on ε, δ.
- Nature knows our strategy, and, after us, chooses a strategy that consists of a probability distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. This strategy can depend on $\varepsilon, \delta, \mathcal{H}$ and also on our strategy, m and $\mathcal{A}$.
- A sample $S \in (\mathcal{X} \times \mathcal{Y})^m$ of size m is drawn according to $\mathcal{D}$.
- The sample S is fed into $\mathcal{A}$ to produce a prediction rule $h_S = \mathcal{A}(S)$. Note that $h_S \in \mathcal{H}$.
- The payoff is $L_{\mathcal{D}}(h_S) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h,(\mathbf{x},y))]$. It is random since S is random and therefore h_S is random.
- Nature does her best to win so we look for learners $\mathcal{A}$ that have a **guaranteed maximal loss** $L_{\mathcal{D}}(h)$ **for any** strategy $\mathcal{D}, f$ that Nature might play.
- To determine if we were successful in the game, we play the game many many times (both us and Nature play the same strategies, just the training samples drawn are different). We count and calculate the probability, over the random draws of training samples S, of the event $\{S \sim \mathcal{D}^m \,|\, L_{\mathcal{D}}(h_S) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \varepsilon\}$. If this probability is found to be larger than $1-\delta$, that is, if the learner $\mathcal{A}$ we chose was Probably Approximately correct with accuracy ε and confidence δ - against Nature's best strategy - *we say that we've been successful (with regards to the parameters ε, δ) and the hypothesis class $\mathcal{H}$.*

Note that in order to calculate the probability of the event $\{S \sim \mathcal{D}^m \,|\, L_{\mathcal{D}}(h_S) \leq \min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h') + \varepsilon\}$ we have to assume a knowledge of the "Oracle quantity" $\min_{h' \in \mathcal{H}} L_{\mathcal{D}}(h')$ - the best possible loss of

any rule in $\mathcal{H}$.

One may ask whether the Agnostic PAC learnability, allowing, for example, more choices for the distribution $\mathcal{D}$, imply PAC-learnability. Not only the answer to this question is positive, but perhaps surprisingly, the inverse claim is also true. In other words, moving to the more general framework of Agnostic-PAC did not change anything, as expressed by the following theorem:

Theorem 4.4.1 Let $\mathcal{X}$ be a sample space and $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ a hypothesis class. Then $\mathcal{H}$ is PAC-Learnable if and only if it is Agnostic-PAC learnable.

4.5 The Fundamental Theorem of Statistical Learning

Recall that in the realizable case, an ERM learner was defined as any $h \in \mathcal{H}$ consistent with the training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ which in turn implied that it minimized the empirical risk. In the current more general case (where the notion of consistency is no more relevant - even the training set does not have to be consistent with itself since the same point may appear with different labels) we redefine ERM as *any* minimizer of the empirical risk.

Definition 4.5.1 — Empirical Risk and ERM Learner in the Agnostic-PAC framework. Let $h : \mathcal{X} \to \mathcal{Y}$ be a prediction rule. We define the ***empirical risk*** of h with respect to the loss function ℓ and the sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ by

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, z_i), \quad z_i = (\mathbf{x}_i, y_i)$$

An ***ERM Learning Algorithm***, $\mathcal{A}_{ERM}$, in the Agnostic-PAC framework is defined as an algorithm that outputs a hypothesis that minimizes the empirical risk:

$$\mathcal{A}_{ERM} : S \mapsto h, \quad h \in \left\{ \operatorname*{argmin}_{h \in \mathcal{H}} L_S(h) \right\}$$

Notice that as before the ERM rule may not be unique. There can be many hypotheses in $\mathcal{H}$ that achieve the minimum $\min_{h \in \mathcal{H}} L_S(h)$.
ERM learners are successful and logical due to the Weak Law of Large Numbers (WLLN) which states that if X_i are a series of $i.i.d$ random variables and $\mu = \mathbb{E}(X_i)$, then

$$\lim_{m \to \infty} \frac{1}{m} \sum_{i=1}^{m} X_i = \mu$$

where the convergence is *in probability*. Namely, for any $\delta > 0$

$$\lim_{m \to \infty} \mathbb{P}\left\{ \left| \frac{1}{m} \sum_{i=1}^{m} X_i - \mu \right| > \delta \right\} = 0$$

which is equivalent to require that for any $\varepsilon, \delta > 0$ there is $m_0 \in \mathbb{N}$ such that for $m > m_0$,

$$\mathbb{P}\left\{ \left| \frac{1}{m} \sum_{i=1}^{m} X_i - \mu \right| > \delta \right\} < \varepsilon.$$

Observe now that for any h we have

- $\mathbb{E}_{\mathcal{D}}[L_S(h)] = L_{\mathcal{D}}(h)$
- By WLLN, when S is $i.i.d$ sample of size m, $\lim_{m\to\infty} L_S(h) = L_{\mathcal{D}}(h)$, in probability
- Therefore for any $\delta > 0$ there is $m_0 \in \mathbb{N}$ such that for $m > m_0$,

$$\mathbb{P}\left\{\left|L_S(h) - L_{\mathcal{D}}(h)\right| > \delta\right\} < \varepsilon\,.$$

- But does this mean that $\text{argmin}_{h\in\mathcal{H}} L_S(h)$ is close to $\text{argmin}_{h\in\mathcal{H}} L_D(h)$?

Although this is a good start, does this imply that $\text{argmin}_{h\in\mathcal{H}}\, L_S\,(h)$ is close to $\text{argmin}_{h\in\mathcal{H}} L_{\mathcal{D}}\,(h)$?

4.5.1 The Fundamental Theorem

Let us reformulate the Fundamental Theorem using Agnostic-PAC learnability and the generalized notion of ERM we just defined.

Theorem 4.5.1 — The Fundamental Theorem of Statistical. Let $\mathcal{H}$ be a hypothesis class of binary classifiers with VC-dimension $d \leq \infty$. Then, $\mathcal{H}$ is **Agnostic-PAC learnable** if and only if $d < \infty$. In this case:

1. There are absolute constants C_1, C_2 such that the sample complexity of $\mathcal{H}$ satisfies

$$C_1 \frac{d + \log(1/\delta)}{\varepsilon^2} \leq m_{\mathcal{H}}(\varepsilon, \delta) \leq C_2 \frac{d + \log(1/\delta)}{\varepsilon^2}$$

2. Furthermore, the upper bound on sample complexity is achieved by the ERM learner.

Note that the "price" we pay for Agnostic PAC learning is that the sample complexity is proportional to $1/\varepsilon^2$ and not to $1/\varepsilon$ as in the PAC Fundamental theorem.

We conclude this chapter with partially going into the proof of the above theorem (4.5.1), and doing so for the qualitative rather than the quantitative part of the theorem. This will help to understand as to how is it that the VC-dimension characterizes learnability, namely, why is that $\mathcal{H}$ **is Agnostic-PAC learnable if and only if** $VCdim(\mathcal{H}) < \infty$. The first part of the theorem is that learning $\mathcal{H}$ is possible if and only if $\mathcal{H}$ is of finite VC-dimension. If it is not of finite VC-dimension then it is impossible to create an Agnostic-PAC learner $\mathcal{A}$ for $\mathcal{H}$ (with accuracy ε and confidence δ) by **any** learning algorithm and using **any** number of training samples.

We have already gained some intuition when discussing the No-Free Lunch theorem (4.2.1). We saw an informal argument that, if there exists $C \subset \mathcal{X}$ that is **shattered** by $\mathcal{H}$, then no learning algorithm can be a Probably Approximately correct learner if it is based on less than $|C|/2$ samples. Now, the statement $VCdim(\mathcal{H}) = \infty$ just means that there are subsets of $\mathcal{X}$ of arbitrary size that are shattered by $\mathcal{H}$ and therefore, no finite sample size will do.

The second part of Lemma 4.5.1, states that if $\mathcal{H}$ is a hypothesis class with $VCdim(\mathcal{H}) = d < \infty$ then $\mathcal{H}$ is Agnostic-PAC learnable as defined in Definition 4.4.4, using any ERM learner. To prove this we will need to define the Uniform Convergence property of hypothesis classes.

4.5.2 Uniform Convergence property

An ERM learner chooses a rule $ERM_{\mathcal{H}}(S)$ which minimizes $L_S(h)$ for the sample S at hand. We hope that the rule $h_S \in ERM_{\mathcal{H}}(S)$, which has minimal empirical risk, will generalize well. Formally, we have to prove that

$$\mathcal{D}^m \left\{ S \in (\mathcal{X} \times \mathcal{Y})^m \,\middle|\, |L_{\mathcal{D}}(h_S) - L_S(h_S)| \leq \varepsilon \right\} \geq 1 - \delta$$

This can only happen if S is a "special" sample - one for which for any $h \in \mathcal{H}$ the empirical risk $L_S(h)$ is pretty close to the generalization loss $L_{\mathcal{D}}(h)$. This is hard to prove. Note that for any $h \in \mathcal{H}$ we have $\mathbb{E}[L_S(h)] = L_{\mathcal{D}}(h)$, so, as we have seen above, by the weak law of large numbers, $L_S(h)$ converges to $L_{\mathcal{D}}(h)$ in probability as the sample size $m \to \infty$. This means that

$$\forall \mathcal{D}\, \forall h \in \mathcal{H}\, \forall \varepsilon, \delta \in (0,1) \quad \exists m_0 \in \mathbb{N} \quad \text{such that} \quad \mathbb{P}\{|L_S(h) - L_{\mathcal{D}}(h)| < \varepsilon\} > 1 - \delta$$

However m_0 depends on both $\mathcal{D}$ and h. We want m_0 that is **uniform** in the distributions $\mathcal{D}$ and the hypotheses $h \in \mathcal{H}$.

Definition 4.5.2 — Uniform Convergence of Function Sequences. A sequence of functions $f_n : X \to \mathbb{R}$ converges uniformly to $f : X \to \mathbb{R}$ if and only if

$$\forall \varepsilon > 0 \quad \exists m_0 \in \mathbb{N} \quad \text{such that} \quad \forall x \in X \; |f_n(x) - f(x)| < \varepsilon$$

Indeed, what we want is to ensure that $L_S(h)$ converges to $L_{\mathcal{D}}(h)$ **uniformly in** $\mathcal{D}$ **and in** $h \in \mathcal{H}$. This leads to the following definition.

Definition 4.5.3 — ε-representative. A training sample S is called ε-representative for $\mathcal{D}, \mathcal{H}, \ell$ if and only if

$$\forall h \in \mathcal{H} \quad |L_S(h) - L_{\mathcal{D}}(h)| < \varepsilon$$

This condition will ensure that minimizing $L_S(h)$ over $h \in \mathcal{H}$ will be close to minimizing $L_D(h)$ over $h \in \mathcal{H}$, which is approximately what we would like to achieve. Specifically, we can show that if we have an ε-representative training set S, then $ERM_{\mathcal{H}}(S)$ will "almost" achieve $\min_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$.

Lemma 4.5.2 Let S be an $\varepsilon/2$-representative sample for $\mathcal{D}, \mathcal{H}, \ell$. Let h_S be any output of $ERM_{\mathcal{H}}(S)$, namely,$h_S \in \operatorname{argmin}_{h \in \mathcal{H}} L_S(h)$. Then

$$L_{\mathcal{D}}(h_S) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon$$

Definition 4.5.4 — Uniform Convergence Property. A hypothesis class $\mathcal{H}$ is said to have the **uniform convergence property** if and only if there exists a function $m_{\mathcal{H}}^{UC} : (0,1)^2 \to \mathbb{N}$ such that for every $\varepsilon, \delta \in (0,1)$ and every distribution $\mathcal{D}$ on $\mathcal{X} \times \mathcal{Y}$

$$\mathcal{D}^m(\{S \in (\mathcal{X} \times \mathcal{Y})^m \,|\, S \text{ is } \varepsilon\text{-representative}\}) \geq 1 - \delta$$

It can then be shown that if a hypothesis class has the uniform convergence property with function $m_{\mathcal{H}}^{UC}$ then $\mathcal{H}$ is Agnostic-PAC learnable with sample complexity $m_{\mathcal{H}}(\varepsilon, \delta) \leq m^{UC}(\varepsilon/2, \delta)$.

Finite VC-Dimension Implies Uniform Convergence Property

So to show Part Two of the fundamental theorem (that ERM is a universal learner), it is enough to show that if $VCdim(\mathcal{H}) < \infty$ then $\mathcal{H}$ has the uniform convergence property. This means showing that for large enough m, that does not depend on $\mathcal{D}$, an $i.i.d$ sample is ε-representative with probability at least $1-\delta$, and that this hold for any possible $\mathcal{D}$.
To achieve uniformity across both $\mathcal{D}$ and $h \in \mathcal{H}$ we define the following functionn $F_m^{\mathcal{D}} : (\mathcal{X} \times \mathcal{Y})^m \to \mathbb{R}$ by

$$F_m^{\mathcal{D}}(S) = \sup_{h \in \mathcal{H}} \left| L_{\mathcal{D}}(h) - L_S(h) \right| \tag{4.3}$$

$F_m^{\mathcal{D}}$ maps a training sample of size m, to a real number measuring its "worse possible confusion" - the maximal difference, over $\mathcal{H}$, between an empirical risk of a hypothesis h and the generalization error of that h. Observe that $F_m^{\mathcal{D}}$ is a function of the random sample S, so it is a random variable, whose distribution depends on the distributions $\mathcal{D}^m$ of training sets of length m.

In essence, we would like to show that with high probability, $F_m^{\mathcal{D}}$ is small. Formally, we would like to show that for every $\varepsilon, \delta \in (0,1)$ there exists $m_{\mathcal{H}}^{UC}(\varepsilon, \delta) \in \mathbb{N}$ such that for every distribution $\mathcal{D}$:

$$\mathcal{D}^m \left\{ F_m^{\mathcal{D}}(S) > \varepsilon \right\} < \delta$$

The Case Of Finite $\mathcal{H}$

To understand the key argument, let us first consider the easier case of finite $\mathcal{H}$ and see how Agnostic-PAC learnability can be proven.

Claim 4.5.3 Let $\varepsilon, \delta \in (0,1)$ then there exists $m_0 \in \mathbb{N}$ such that for any $m > m_0$ it holds that

$$\forall \mathcal{D} \text{ over } \mathcal{X} \times \mathcal{Y} \quad \mathcal{D}^m \left\{ F_m^{\mathcal{D}}(S) > \varepsilon \right\} \leq \delta$$

Proof. Directly by definition then

$$\begin{aligned}
\mathcal{D}^m \left\{ F_m^{\mathcal{D}}(S) > \varepsilon \right\} &\overset{def.}{=} \mathcal{D}^m \left\{ S \,\middle|\, \exists h \in \mathcal{H}, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon \right\} \\
&\overset{union\ bound}{\leq} \sum_{h \in \mathcal{H}} \mathcal{D}^m \left\{ S \,\middle|\, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon \right\} \\
&\leq |\mathcal{H}| \cdot \max_{h \in \mathcal{H}} \mathcal{D}^m \left\{ S \,\middle|\, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon \right\}
\end{aligned}$$

We thus need to bound $\mathcal{D}^m \left\{ S \,\middle|\, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon \right\}$ uniformly in $\mathcal{D}$ and h. By the weak law of large numbers (WLLN), since $L_S(h)$ is an empirical mean of $i.i.d$ random variables with expected value $L_{\mathcal{D}}(h)$, we know that

$$\forall \varepsilon > 0 \quad \mathcal{D}^m \left\{ \left| L_S(h) - L_{\mathcal{D}}(h) \right| > \varepsilon \right\} \overset{m \to \infty}{\longrightarrow} 0$$

This is not sufficient as we want the bound to not depend on $\mathcal{D}, h$. What we need is a known as a **concentration of measure** inequality: a way to bounds the distance between the empirical mean

and the expected value. Indeed, recall Hoeffding's Inequality: Let $\theta_1, \dots \theta_m$ be a sequence of $i.i.d$ random variables and assume that for all i, $\mathbb{E}[\theta_i] = \mu$ and $\mathbb{P}\{a \leq \theta_i \leq b\} = 1$. Then:

$$\forall \varepsilon > 0 \quad \mathbb{P}\left\{\left|\frac{1}{m}\sum_{i=1}^{m}\theta_i - \mu\right| > \varepsilon\right\} \leq 2\exp\left(-2\frac{m\varepsilon^2}{(b-a)^2}\right)$$

Therefore, let us define $\theta_i = \ell(h,(\mathbf{x}_i, y_i))$. Observe that $L_{\mathcal{D}}(h) = \mathbb{E}[\theta_i]$, where the expectation is with respect to $\mathcal{D}$, and that $L_S(h) = \frac{1}{m}\sum_{i=1}^{m}\theta_i$. As such we get that

$$\mathcal{D}^m\left\{S \,\middle|\, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\right\} \leq 2\exp\left(-2m\varepsilon^2\right)$$
$$\Downarrow$$
$$|\mathcal{H}| \cdot \max_{h\in\mathcal{H}} \mathcal{D}^m\left\{S \,\middle|\, |L_S(h) - L_{\mathcal{D}}(h)| > \varepsilon\right\} \leq 2|\mathcal{H}|\exp\left(-2m\varepsilon^2\right)$$

By choosing that $m \geq \frac{\log(2|\mathcal{H}|/\delta)}{2\varepsilon^2}$ we get that:

$$\mathcal{D}^m\left\{F_m^{\mathcal{D}}(S) > \varepsilon\right\} \leq 2|\mathcal{H}|\exp\left(-2m\varepsilon^2\right) \leq \delta$$

■

The Case Of Infinite $\mathcal{H}$

Unfortunately, we are not apply to apply the union bound over an inifinite number of hypotheses. Instead, recall that for every finite $C \subseteq \mathcal{X}$, we write $\mathcal{H}_C$ for the hypotheses in $\mathcal{H}$, all restricted to C. The key to the proof is to understand **how fast** can the restriction $\mathcal{H}_C$ grow with $|C|$. If $|C| \leq VCdim(\mathcal{H})$ it could be that $\mathcal{H}$ shatters $|C|$ and therefore it could be that $|\mathcal{H}_C| = 2^{|C|}$. However, if $|C| > VCdim(\mathcal{H})$ it can't be - by definition - that $|\mathcal{H}_C| = 2^{|C|}$. So the question is how large can $|\mathcal{H}_C|$ be.

Definition 4.5.5 For a hypothesis class $\mathcal{H}$ Define $\tau\mathcal{H}(m)$ by

$$\tau_{\mathcal{H}}(m) = \max\left\{|\mathcal{H}_C| \,\middle|\, C \subset \mathcal{X}, |C| = m\right\}$$

This definition is a combinatorial property of $\mathcal{H}$: the maximal number of functions that can be obtained by restricting $\mathcal{H}$ to any subset of size m. The larger and more complicated $\mathcal{H}$, the larger we can expect $\tau_{\mathcal{H}}(m)$ to be. In other words, $\tau_{\mathcal{H}}(m)$ measures how fast - at most - $\mathcal{H}_C$ can grow with $|C|$. For example, we saw that if $VCdim(\mathcal{H}) = \infty$, then $\tau_{\mathcal{H}}(m) = 2^m$, namely, $\mathcal{H}_C$ can grow exponentially in $|C|$.

Definition 4.5.6 Let $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$. Suppose there exist $m_0 \in \mathbb{N}$, $b > 0$ and $\beta > 0$ such that: for all

$$\forall m > m_0 \quad \tau_{\mathcal{H}}(m) \leq b \cdot m^{\beta}$$

Then we say that $\mathcal{H}_C$ grows **polynomially** in $|C|$.

So the proof that a finite VC-dimension implies the uniform convergence is based on two parts:

1. If $|\mathcal{H}_C|$ grows polynomially in $|C|$, then $\mathcal{H}$ has the uniform convergence property. Hence it Agnostic-PAC learnable using the ERM rule.
2. If $VCdim(\mathcal{H}) < \infty$, then $|\mathcal{H}_C|$ grows polynomially in $|C|$.

Claim 4.5.4 Let $\mathcal{H}$ be a hypothesis class such that $|\mathcal{H}_C|$ grows polynomially in $|C|$ then $\mathcal{H}$ has the uniform convergence property.

Proof. Recall that we would like to show that

$$\mathcal{D}^m\left(\left\{F_m^{\mathcal{D}}(S) > \varepsilon\right\}\right) < \delta$$

uniformly in $\mathcal{D}$. Since $F_m^{\mathcal{D}}$ is a non-negative random variable, we consider Markov's inequality. We would like to define a sequence of numbers α_m that will depend on $\mathcal{H}$ but *not* on the distribution $\mathcal{D}$, for which it holds that that

$$\mathbb{E}_{\mathcal{D}^m}\left[F_m^{\mathcal{D}}(S)\right] \leq \alpha_m \tag{4.4}$$

By doing so we will bound $\mathbb{E}_{\mathcal{D}^m}[F_m^{\mathcal{D}}(S)]$ uniformly across $\mathcal{D}$. If we are successful in doing so then

$$\mathbb{P}_{\mathcal{D}^m}\left\{\sup_{h\in\mathcal{H}}\left|L_{\mathcal{D}}(h) - L_S(h)\right| > \varepsilon\right\} = \mathbb{P}_{\mathcal{D}^m}\left\{F_m^{\mathcal{D}}(S) > \varepsilon\right\} \leq \frac{\mathbb{E}_{\mathcal{D}^m}[F_m^{\mathcal{D}}(S)]}{\varepsilon} \leq \frac{\alpha_m}{\varepsilon}.$$

Which means that with probability at least $1 - \alpha_m/\varepsilon$, a training set of length m is ε-representative. Therefore, we managed to achieve uniformity across both $h \in \mathcal{H}$ (by using $F_m^{\mathcal{D}}$, a supremum over h) and over $\mathcal{D}$ (by bounding the expected value of $F_m^{\mathcal{D}}$ independently of $\mathcal{D}$).

Note, that is we are able to find such as sequence α_m that decreases to 0, then for any ε, δ we can set m_0 to be such that for all $m > m_0$, $\alpha_m/\varepsilon < \delta$. This would imply that $\mathcal{H}$ has the uniform convergence property. To find such a sequence α_m we use the following lemma

Lemma 4.5.5 Let $F_m^{\mathcal{D}}$ be as in (4.3). Then independently of $\mathcal{D}$

$$\mathbb{E}_{\mathcal{D}^m}\left[F_m^{\mathcal{D}}(S)\right] \leq \mathcal{O}\left(\frac{\sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\sqrt{2m}}\right) + o(m)$$

Since we assumed that $|\mathcal{H}_C|$ grows polynomially in $|C|$, we have for all $m > m_0$ (for some m_0) that $\tau_{\mathcal{H}}(m) \leq b \cdot m^{\beta}$ for some $b, \beta > 0$. Hence,

$$\mathbb{E}_{\mathcal{D}^m}[F_m^{\mathcal{D}}(S)] \leq O\left(\frac{\sqrt{\beta \cdot \log(2m)}}{\sqrt{2m}}\right) + o(m) \searrow 0$$

and therefore if $|\mathcal{H}_C|$ grows polynomially in $|C|$ then $\mathcal{H}$ has the uniform convergence property. ■

Claim 4.5.6 Let $\mathcal{H}$ be a hypothesis class with a finite VC-dimension. Then $|\mathcal{H}_C|$ grows polynomially in $|C|$

Proof. By definition, if $m \leq VCdim(\mathcal{H})$ then there exists a set $C \subset \mathcal{X}$, of size m, which is shattered by $\mathcal{H}$. This means that if $m \leq VCdim(\mathcal{H})$ then $\tau_{\mathcal{H}}(m) = 2^m$. It can be shown, see Ex.7, that if $m > VCdim(\mathcal{H})$ then $\tau_{\mathcal{H}}(m) \leq (em/d)^d$. Therefore, while $\tau_{\mathcal{H}}(m)$ grows exponentially in m for $m \leq VCdim(\mathcal{H})$, it only grows **polynomially** in m for $m > VCdim(\mathcal{H})$. ■

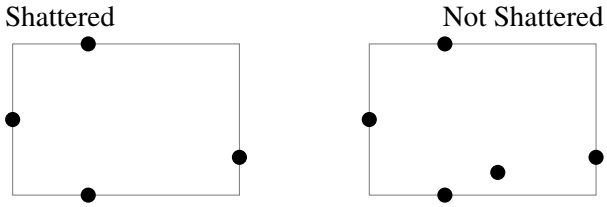
So, that was a taste of the proof of the second part of the fundamental theorem. We proved everything formally, except Lemma 2. This lemma is indeed deep and meaningful: it bounds the expected value of the "worse possible deviation" between empirical risk and generalization error, $\sup_{h\in\mathcal{H}}\left|L_{\mathcal{D}}(h)-L_S(h)\right|$, over a random choice of training sample, uniformly in $\mathcal{D}$. The bound uses $\tau_{\mathcal{H}}(m)$, which bounds how fast the size of a restriction $\mathcal{H}_C$ can grow with $|C|$.

4.6 Summary and Exercises

1. The ***Axis-aligned rectangles*** hypothesis class over the sample space $\mathcal{X}=\mathbb{R}^2$ is defined as:

$$\mathcal{H}=\left\{h_{(a_1,a_2,b_1,b_2)}:a_1<a_2\wedge b_1<b_2\right\}\quad h_{(a_1,a_2,b_1,b_2)}(x_1,x_2)=\mathbb{1}\left[x_1\in[a_1,a_2]\wedge x_2\in[b_1,b_2]\right]$$

   Note that every function in this hypothesis class defines a finite rectangle aligned with the x and y axes $\mathbb{R}^2$. Show that no set of 5 points can be shattered by the Axis-aligned rectangles class while any 4 points located on 4 different edges (away from the corners) of any given rectangle can:



   Hint: note that the 3 points (x_k,y_k), (x_i,y_i), and $(x_{k'},y_{k'})$ can not be shattered if $x_k\le x_i\le x_{k'}$ and $y_k\le y_i\le y_{k'}$.
2. Consider the case of a finite hypothesis class.
   - Show that the VC dimension of a finite $\mathcal{H}$ is at most $\log_2(|\mathcal{H}|)$.
   - Show that, over the same sample space, $\mathcal{X}$, VC dimensions can take any value between 1 and $\log_2(|\mathcal{H}|)$ even for hypothesis classes of the same size: Given $n\in\mathbb{N}$, find a sample space, $\mathcal{X}$, and a hypothesis class, $\mathcal{H}^{(n)}$ with $VCdim(\mathcal{H}^{(n)})=\log_2(|\mathcal{H}^{(n)}|)=n$. Then, for each $k=1..n-1$ construct a hypothesis class, $\mathcal{H}^{(k)}$, with $|\mathcal{H}^{(k)}|=|\mathcal{H}^{(n)}|$ but with $VCdim(\mathcal{H}^{(k)})=k$.
3. Let $\mathcal{X}=\mathbb{R}^d$. The ***hypothesis class*** of half-spaces through the origin is defined as

$$\mathcal{H}=\left\{\mathbf{x}\mapsto sign(\mathbf{x}^\top\mathbf{w}):\mathbf{w}\in\mathbb{R}^d\right\}$$

   - Show that $\{\mathbf{e}_1,\ldots,\mathbf{e}_d\}$ is shattered by $\mathcal{H}$.
   - Show that any $d+1$ points cannot be shattered (consider the standard basis vectors).
   - What is $VCdim(\mathcal{H})$?
4. Let $|\mathcal{X}|=\infty$. Let $\mathcal{H}\subset\{\pm1\}^{\mathcal{X}}$ with VCdim($\mathcal{H}$)<∞. Let $\mathcal{H}'$ be the complement hypothesis class, namely, $\mathcal{H}'\equiv\{\pm1\}^{\mathcal{X}}\setminus\mathcal{H}$. Let $C\subset\mathcal{X}$, $|C|<\infty$, be *any* finite subset of $\mathcal{X}$. Show that $\mathcal{H}'$ shatters C.
5. Let $\mathcal{X}$ be a sample space, $\mathcal{H}\subset\mathcal{Y}^{\mathcal{X}}$ a hypothesis class, and let ℓ_{0-1} be the $0-1$ loss function. Let $\varepsilon,\delta\in(0,1)$. Assume that $h\in\mathcal{H}$ is an Agnostic Probably Approximately correct learner with accuracy ε and confidence δ, with respect to ℓ_{0-1}, $\mathcal{H}$. Show that $\mathcal{A}$ is a Probably Approximately Correct learner (with accuracy ε and confidence δ).

6. Prove Lemma 4.5.2: for S an $\varepsilon/2$-representative sample for $\mathcal{D}, \mathcal{H}, \ell$ any ERM ourput h_S satisfies that
$$L_{\mathcal{D}}(h_S) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \varepsilon$$
7. Let $\mathcal{H}$ be some hypothesis class of functions $\mathcal{X} \to \{\pm 1\}$ with finite VC-dimension. If $m > VCdim(\mathcal{H})$ then $\tau_{\mathcal{H}}(m) \leq (em/d)^d$.
8. Let $\tilde{\mathcal{D}}$ be a distribution on $\mathcal{X}$ alone, and let $f : \mathcal{X} \to \mathcal{Y}$ be a labeling function, $\mathcal{Y} = \{\pm 1\}$. Construct an equivalent joint distribution $\mathcal{D}(\mathbf{x}, y)$ on $\mathcal{X} \times \mathcal{Y}$, assuming no-noise. Verify the answer by calculating the distribution for the pair $(\mathbf{x}, f(\mathbf{x}))$ and for the pair $(\mathbf{x}, -f(\mathbf{x}))$.
9. Consider again $\tilde{\mathcal{D}}(\mathbf{x})$ and $\mathcal{D}(\mathbf{x}, y)$ defined in Ex.8. Verify that Equation 4.1 gives, for $\tilde{\mathcal{D}}(\mathbf{x})$, the same misclassification loss as Equation 4.2 gives, for $\mathcal{D}(\mathbf{x}, y)$.

5. Ensemble Methods

In previous chapters we discussed different classification algorithms, each of which implements a different learning principle for choosing the learning rule $h_s \in \mathcal{H}$. In this chapter we will not cover new learning algorithms per se, but rather consider several general "meta-algorithms" for the creation of *ensembles* of classifiers. These collection of classifier can be applied to any existing learning algorithm and improve its performance. We will learn about the three B's: *Bootstrapping*, *Bagging* and *Boosting* and understand how they give us better control over the *Bias-Variance Trade-off*. These powerful techniques are broadly used and applicable for many difficult problems even outside the context of the meta-algorithms presented in this chapter

5.1 Bias-Variance Trade-off

Throughout the book we have stated informally that the "larger" or "more complicated" our chosen hypothesis class, typically our learner will have lower **bias** and higher **variance**. We said informally that bias is part of the generalization error that is incurred by the "best" hypothesis in $\mathcal{H}$. If we think of an unknown labeling function f chosen by nature, then bias measures how well the unknown labeling function f can be decried by the "closest" hypothesis in $\mathcal{H}$. Intuitively, the larger $\mathcal{H}$, the more expressive power it has to describe more complicated functions f - hence a lower bias. We also said informally that variance is the part of the generalization error that is incurred by the fact that the training sample is random, hence our chosen rule h_s is also random. The larger $\mathcal{H}$ will be, the more freedom our learning algorithm has to "chase" random fluctuations in the training sample, which do not represent the underlying labeling we are trying to learn.

R The variance part can be further broken down into two parts - one part comes from randomness in the choice of training samples while another part comes from the measurement noise or noise in the labels. For simplicity we will represent the variance as a single component.

As such, the bias-variance **tradeoff** is that: the more complicated the model, the smaller the bias and the larger the variance. Informally, the generalization error is some combination of the two. In cases where we can tune the model complexity (that is, the size/complexity of the hypothesis class) we would like to look for the "sweet spot" of a model that has "just the right amount of complexity".

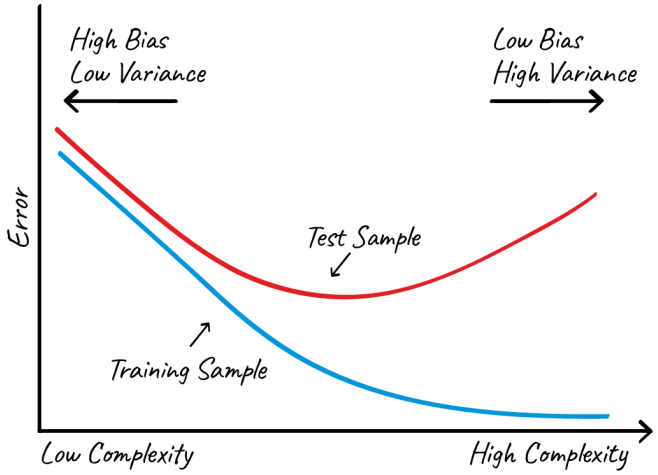


Figure 5.1: The bias-variance tradeoff: *Train- vs. test errors as function of complexity of h*

The methods describe below allow us to escape the tradeoff in some sense. They enable the reduction of the bias or variance of a learner without substantially increasing the other.

5.1.1 Generalization Error Decomposition

We have already encountered the generalization error decomposition when discussing linear regression problems. There we have shown how we can decompose the MSE into the bias and variance components (2.18). Next, let us revisit the decomposition but for some general loss function. Let $h^* = \text{argmin}_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ and $h_S = \mathcal{A}(S)$ be the output of a learning algorithm, then we can decompose the generalization error of the hypothesis returned by the learner as follows:

$$L_{\mathcal{D}}(h_S) = \underbrace{L_{\mathcal{D}}(h^*)}_{\varepsilon_{approximation}} + \underbrace{L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*)}_{\varepsilon_{estimation}} \tag{5.1}$$

- The **approximation error** is $L_{\mathcal{D}}(h^*)$. Namely, the error of the hypothesis $h \in \mathcal{H}$ achieving the lowest generalization error. This term does not depend at all on our training sample and its size m. It depends only on the selection of $\mathcal{H}$. As we expand the hypothesis class to become richer we might find a better hypothesis for explaining the data. This error is what we already know as the **bias** error, induced by restricting the hypothesis class.
- The **estimation error** is $L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*)$. Namely, it is the difference between the generalization error achieved by the selected hypothesis and the best hypothesis in $\mathcal{H}$. This term depends on the training set and its size. This error is what we already know as the **variance** error.

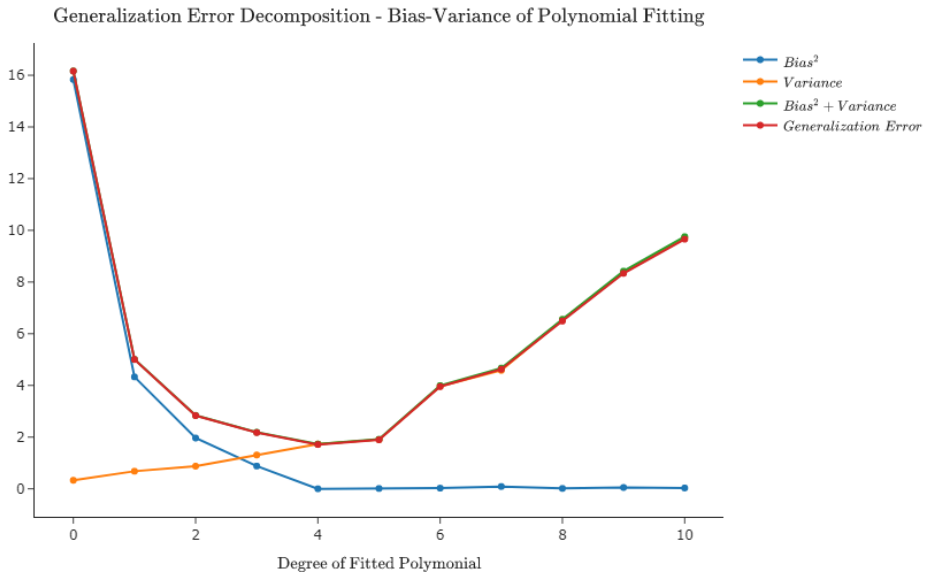


Figure 5.2: Bias-Variance Tradeoff Graph: *for polynomial fitting of true polynomial degree* 4. *Chapter 5 Examples - Source Code*

5.2 Ensemble/Committee Methods

> "A collective wisdom of many is likely more accurate than any one." — Aristotle, in *Politics*, circa 300BC

Before exploring specific ensemble methods, let us analyze some mathematical properties of a committee based decision. This will provide insights into how committee based methods manage to improve generalization. Consider a committee of T members, which has to make a "yes"/"no" decision. Each member casts a vote, which with probability p_i being correct and probability $1-p_i$ being wrong. Let us further assume for simplicity that all members are "equally wise", so that p is the same for all members. After all members vote, the committee's decision is simply the majority vote. For this setup we can ask questions such as:

- What is the probability of the committee deciding the right decision?
- What would a typical decision be? and how consistent is it?
- How does the number of members in the committee influence the measures above?
- If committee members are not independent from one another, and influence each other's decisions, how does it influence the measures above?

We begin answering these questions theoretically starting with the probability of a committee deciding the right decision.

Lemma 5.2.1 Let $X_1,\ldots,X_T \overset{i.i.d}{\sim} Ber(p)$ taking values in $\{\pm 1\}$ and denote $X=\sum X_i$. The probability of the committee making the correct decision is $\mathbb{P}(X>0)$.

Proof. As the committee decides by a majority vote then the probability of the committee deciding right is the same as the probability of having more members deciding right than members deciding

wrong:

$$\mathbb{P}(\text{Committee decides right}) = \mathbb{P}(|\text{Decided right}| > |\text{Decided wrong}|)$$

W.o.l.g, suppose the true answer is $+1$. As each random variable takes a value in $\{\pm\}$ we could express the collective vote as $X = sign(\sum X_i)$. If the committee decided right, then there are more members that voted right and $\sum X_i > 0$ which means that $X = 1$. On the other hand, if the committee decided wrong, then more members voted wrong and $\sum X_i <= 0$ which means that $X = -1$. So, we conclude that:

$$\mathbb{P}(\text{Committee decides right}) = \mathbb{P}(X > 0)$$

■

Lemma 5.2.2 Let $X_1, \ldots, X_T \overset{i.i.d}{\sim} Ber(p)$ taking values in $\{\pm 1\}$ with $p > 0.5$ and denote $X = \sum X_i$. The probability of the committee deciding correctly is bounded below by $1 - \exp\left(-\frac{T}{2p}\left(p - \frac{1}{2}\right)^2\right)$.

Proof. W.l.o.g let us assume that the correct answer is $+1$ and denote $X = \sum_{i=1}^{T} X_i$. We are therefore interested in bounding $\mathbb{P}(X > 0)$ from below. We will achieve this by bounding $\mathbb{P}(X \leq 0)$ from above. Notice that for any $a > 0$ it holds that:

$$\mathbb{P}(X \leq 0) = \mathbb{P}(-aX \geq 0) = \mathbb{P}\left(e^{-aX} \geq e^0\right)$$

Now, using Markov's inequality

$$\mathbb{P}(X \leq 0) \leq \mathbb{E}\left[e^{-aX}\right] = \mathbb{E}\left[e^{-a\sum_i X_i}\right] \overset{iid}{=} \mathbb{E}\left[e^{-aX_1}\right]^T$$

Notice that as $X_1 \sim Ber(p)$ over $\{\pm 1\}$ it holds that:

$$\mathbb{E}\left[e^{-aX_1}\right] = pe^{-a} + (1-p)e^a = e^a\left(1 - p + pe^{-2a}\right) \leq e^{a-p+pe^{-2a}}$$

where the last inequality is because $1 + x \leq e^x$. Next, we use the inequality $x\ln(x) \geq \frac{x^2}{2} - \frac{1}{2} \quad x \in (0,1)$. For a selection of $a = \frac{1}{2}ln(2p)$ which is positive for $p > 0.5$ we get that:

$$\begin{array}{rclcl}
\mathbb{P}(X \leq 0) & \leq & \mathbb{E}\left[e^{-aX_1}\right]^T & \leq & \exp\left(T\left(a - p + pe^{-2a}\right)\right) \\
& = & \exp\left(T\left(\frac{1}{2}ln(2p) - p + \frac{1}{2}\right)\right) & = & \exp\left(Tp\left(-\frac{1}{2p}ln\left(\frac{1}{2p}\right) - 1 + \frac{1}{2p}\right)\right) \\
& \leq & \exp\left(Tp\left(\frac{1}{2} - \frac{1}{2(2p)^2} - 1 + \frac{1}{2p}\right)\right) & = & \exp\left(-\frac{Tp}{2}\left(\frac{1}{4p^2} - \frac{1}{p} + 1\right)\right) \\
& = & \exp\left(-\frac{Tp}{2}\left(\frac{1}{2p} - 1\right)^2\right) & = & \exp\left(-\frac{T}{2p}\left(p - \frac{1}{2}\right)^2\right)
\end{array}$$

Finally, we conclude that:

$$\mathbb{P}(X > 0) = 1 - \mathbb{P}(X \leq 0) \geq 1 - \exp\left(-\frac{T}{2p}\left(p - \frac{1}{2}\right)^2\right)$$

■

Therefore, it turns out that if each member is typically right ($p > 0.5$) then the probability that the committee is right is higher than any individual member. In addition, the probability of the committee being wrong decays with a rate of $\mathcal{O}\left(e^{-T}\right)$. Relating this to our learning scheme, it means that the confidence $1-\delta$ in our prediction increases exponentially as T increases.

Based on (5.2.1) and (5.2.2) we can now simulate different scenarios for different values of T and p and see how committees behave. Figure 5.3 shows the theoretical bound achieved above and the empirical results for committees if increasing size and for different values of p. We can see that:

- The larger the committee size T the higher the probability of the committee being correct.
- The larger the probability of each committee member of being correct p, the fewer committee members are needed for the committee to be correct with probability of 1.

We also see that empirical results agree with the theoretical lower bound devised above.

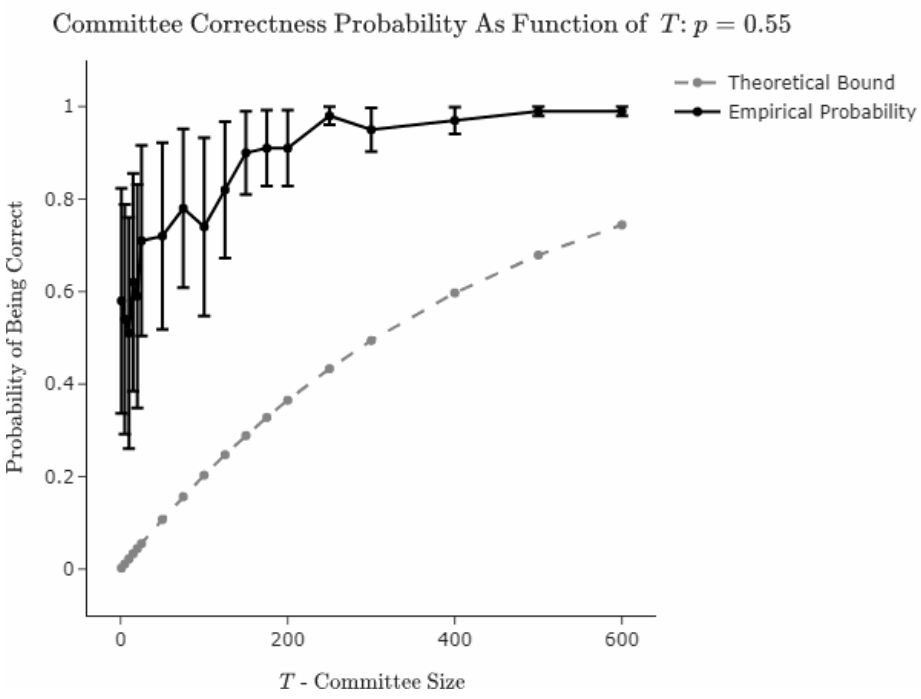


Figure 5.3: Committee Decision - Correctness Probability: *Theoretical bounds and empirical results as function of T, p.* *Chapter 5 Examples - Source Code*

5.2.1 Uncorrelated Predictors

Next, let us calculate what is a typical decision ($\mathbb{E}(X)$) and how consistent is it ($Var(X)$)?

> **Exercise 5.1** Let $X_1, \ldots, X_T \overset{iid}{\sim} Ber(p)$ taking values of $\{\pm 1\}$ with $p > 0.5$. What is the expectation and variance of $X = \frac{1}{T}\sum_{i=1}^{T} X_i$?

Solution: We begin with calculating the expectation and variance of each committee member:

$$\begin{aligned}\mathbb{E}[X_i] &= 1\cdot\mathbb{P}(X_i=1)+(-1)\cdot\mathbb{P}(X_i=-1)\\ &= 2p-1\end{aligned}$$

$$\begin{aligned}Var(X_i) &= \mathbb{E}\left[(X_i-\mathbb{E}[X_i])^2\right]\\ &= p(1-(2p-1))^2+(1-p)(-1-(2p-1))^2\\ &= 4p(1-p)^2+4p^2(1-p)\\ &= 4p(1-p)\end{aligned}$$

Then, for X:

$$\begin{aligned}\mathbb{E}[X] &= \tfrac{1}{T}\textstyle\sum_i \mathbb{E}[X_i] = 2p-1\\ Var(X) &= \tfrac{1}{T^2}Var\left(\textstyle\sum_i X_i\right) \overset{iid}{=} \tfrac{1}{T^2}\textstyle\sum_i Var(X_i) = \tfrac{4}{T}p(1-p)\end{aligned}$$

Therefore, when using a committee of independent members the expectation of decision remains the same while decreasing the variance at a rate of $\mathcal{O}\left(\frac{1}{T}\right)$. In other word, we are able to keep the same accuracy while increasing the confidence.

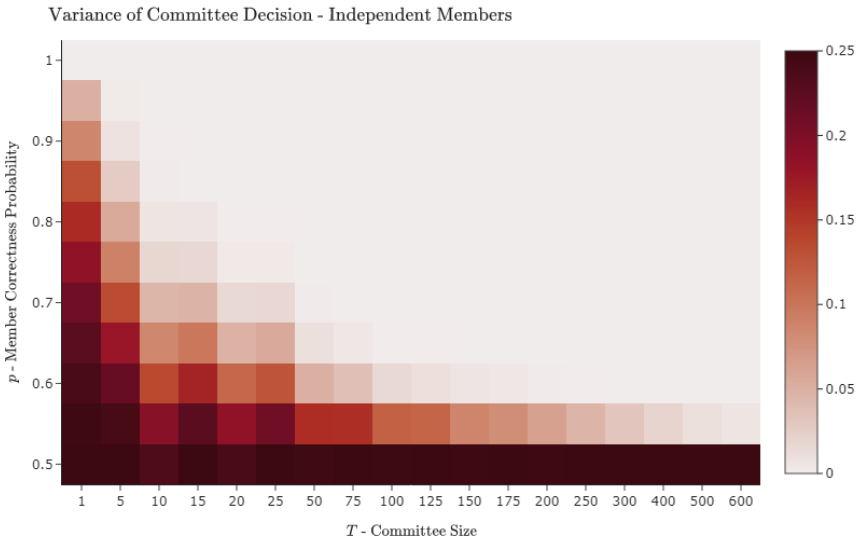


Figure 5.4: Variance of Committee Decision: *with independent members, as function of T, p.*
Chapter 5 Examples - Source Code

5.2.2 Correlated Predictors

In practice, however, committee members rarely vote independently. So let us assume that each two members are correlated with equal correlation $\rho \in [0,1]$.

Lemma 5.2.3 Let $X_1,\ldots,X_T$ be a set of identically-distributed real-valued random variables such that: $Var(X_i)=\sigma^2$ and $corr(X_i,X_j)=\rho$, $i\neq j$. The variance in the commitee's decision is given by $\rho\sigma^2+\frac{1}{T}(1-\rho)\sigma^2$.

Proof. As X is the average of T identically distributed random variables:

$$Var(X) = Var\left(\frac{1}{T}\sum_i X_i\right) = \frac{1}{T^2}\left[\sum_i Var(X_i) + 2\sum_{i<j} Cov(X_i, X_j)\right]$$

Recall that the correlation between two random variables is defined as:

$$corr(A,B) := \frac{Cov(A,B)}{\sqrt{Var(A)\,Var(B)}}$$

and therefore for any $i \neq j$:

$$Cov(X_i, X_j) = corr(X_i, X_j)\sqrt{Var(X_i)\,Var(X_j)} = \rho\sigma^2$$

Plugging this back into the variance:

$$Var(X) = \frac{1}{T^2}\left[T\sigma^2 + 2\binom{T}{2}\rho\sigma^2\right] = \frac{\sigma^2}{T} + \left(1 - \frac{1}{T}\right)\rho\sigma^2 = \rho\sigma^2 + \frac{1}{T}(1-\rho)\sigma^2$$

■

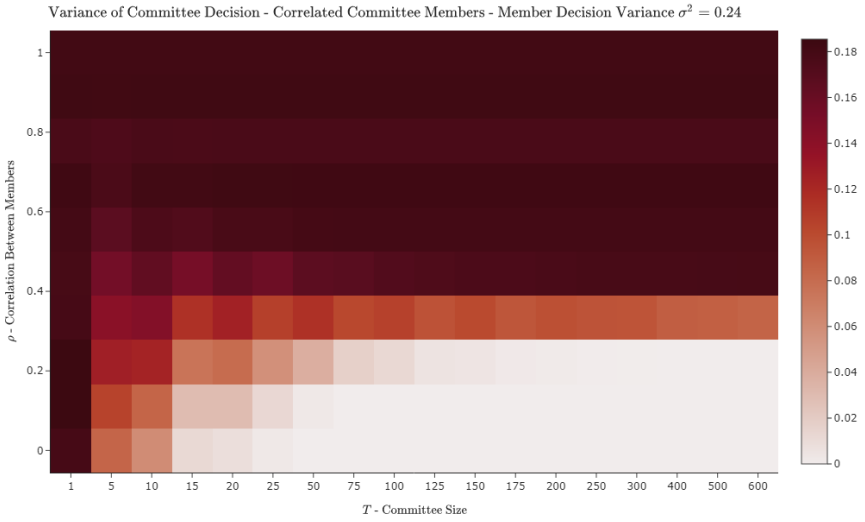


Figure 5.5: Variance of Committee Decision: *with correlated members, as function of* T, ρ. *Chapter 5 Examples - Source Code*

All together, we have seen analytically and quantitatively that given a committee of members deciding by majority vote, where decisions of members are correlated with correlation ρ and each member is correct with probability p:

- If $p > 0.5$ the decision made by the committee improves with T in two ways: higher probability of being right, and its decision will be more consistent.
- If $\rho > 0$ then increasing T will increase the probability of the decision made by the committee being right up to a certain point.

5.2.3 Committee Methods In Machine Learning

Let us apply these ideas to machine learning applications. Suppose we have T training samples $S_1,\ldots,S_T$ of size m chosen independently from $\mathcal{X}$ according to some distribution $\mathcal{D}$. Let $\mathcal{A}$ be a learning algorithm and train it on each of the training samples, to obtain $h_{S_1},\ldots,h_{S_T}$. Let us consider $h_{S_t}(x)$ for some $x\in\mathcal{X}$. As $S_t \overset{i.i.d}{\sim} \mathcal{D}^m$ we can think of the training set as a random variable. This means that the also h_{S_t} obtained by training $\mathcal{A}$ over S_t is a random variable. Lastly, it means that we can think of the prediction $h_{S_t}(x)$ as a random variable, which has some distribution. In addition, notice that as the training samples are chosen independently, predictions of different h_{S_t} over x are also independent random variables. So, if we use $h_{S_1},\ldots,h_{S_T}$ in a committee we have the situation described above. The generalization loss will be reduced as T grows as the variance of the prediction will decrease as a rate of $1/T$.

However, in batch learning we don't have T training samples, but rather just one, so how would we accuire such different hypotheses? We cannot train $\mathcal{A}$ over S multiple times as we will get identical predictions, which are perfectly correlated. Instead, what we would like to do is to create T training samples from the original one. If we could mimic fresh independent draws of new traning samples of size m according to $\mathcal{D}$.

Definition 5.2.1 — Committee Methods. Let $\mathcal{A}$ be some learner predicting labels in $\{\pm 1\}$. A committee method over $\mathcal{A}$ is the function:

$$h(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right)$$

That is, in committee methods (or ensembles) we take an existing "base" learner and apply it ot a sequence of T training samples. For the remaining of this chapter we will introduce two very different ideas for building the committee member rules.

5.3 Bagging

5.3.1 The Bootstrap

For the first committee method we begin with introducing a concept from statistics: **The Bootstrap**. This is one of the most groundbreaking ideas of statistics in the 20th century, where we create new "artificial" training samples from the one training sample at hand.

Given a training sample $S=\{(\mathbf{x}_i,y_i)\}_{i=1}^m$ we are going to construct a new training sample, called a *bootstrap sample* S^{*1}. We sample m times from S **with replacement** and denote this "new" sample by:

$$S^{*1} = \left\{\left(\mathbf{x}_i^{*1}, y_i^{*1}\right)\right\}_{i=1}^m$$

Of course, since we sampled from S with replacements, there might be repeated samples in S^{*1}, even if S itself had no repeated samples. Now we can repeat this process B times, obtaining B training samples, each of length m: $S^{*1},\ldots,S^{*B}$. The samples in the b-th training sample will be denoted

$$S^{*b} = \left\{\left(\mathbf{x}_i^{*b}, y_i^{*b}\right)\right\}_{i=1}^m$$

Using the newly created bootstrap samples we can now train our base learner $\mathcal{A}$ over each one separately, obtain B prediction rules and form an ensemble. But so how is it that bootstrap actually works? Assume the samples in our learning problem are $i.i.d$ samples from an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. We are hoping that each Bootstrap from S somehow behaves like a fresh $i.i.d$ sample from $\mathcal{D}$ itself. Given a training sample S we can define the **empirical distribution** $\widehat{\mathcal{D}}_S$ induced by S on $\mathcal{X} \times \mathcal{Y}$ as the following probability distribution on $\mathcal{X} \times \mathcal{Y}$: for a subset $C \subset \mathcal{X} \times \mathcal{Y}$, define:

$$\widehat{\mathcal{D}}_S\left((X,Y) = (x,y)\right) := \begin{cases} \frac{1}{m} & (x,y) \in S \\ 0 & (x,y) \notin S \end{cases}$$

or equivalently, for any $C \subset \mathcal{X} \times \mathcal{Y}$:

$$\widehat{\mathcal{D}}_S(C) := \frac{|C \cap S|}{m}$$

where for simplicity we assume all samples in S are unique. Observe that this is equivalent to putting a probability mass of $1/m$ on each of the points of S, and zero mass on all other points in $\mathcal{X} \times \mathcal{Y}$. Observe that a bootstrap sample S^{*b} is just an $i.i.d$ draw of m points from the **empirical distribution** $\widehat{\mathcal{D}}_S$ induced by the one training sample we have, S. As m grows, namely as S becomes larger, the empirical distribution $\widehat{\mathcal{D}}_S$ converges in distribution to $\mathcal{D}$. The idea behind the bootstrap is that, if $\widehat{\mathcal{D}}_S$ is not so different from $\mathcal{D}$, then m $i.i.d$ draws from $\widehat{\mathcal{D}}_S$ is a good approximation to m $i.i.d$ draws from $\mathcal{D}$.

One way to see the convergence of the empirical distribution to the underlying distribution is on the real line. Figure 5.6 shows the empirical CDF of samples drawn from a standard normal distribution and compares it with the theoretical CDF of the distribution. As the number of samples increases the empirical CDF converges to the CDF of the standard distribution.

5.3.2 Bagging

The idea of Bootstrap samples can be used in any scenario where we wish to create new artificial samples from our only training sample S. It has many uses throughout machine learning, statistics and data science. **Bagging** is a nickname of one such usage where we use Boostrap, to improve the accuracy of an existing supervised machine learning algorithm.

We start with a "base" learning algorithm $\mathcal{A}$ and a training sample S. We then form T bootstrap training samples, $S^{*1}, \ldots, S^{*T}$, each of size m. We then train our learner **separately** on each of the T bootstrap training samples. We form the committee $h_{S^{*1}}, \ldots, h_{S^{*T}}$ and store all T trained models. When we need to classify a new test sample $x \in \mathcal{X}$, we run x through all the rules and classify using the majority vote of the committee,

$$h_{bag}(x) := sign\left(\sum h_{S^{*t}}(x)\right)$$

For example, if we run Bagging on top of the Decision Tree classifier, we'll obtain a committee of decision trees:

Note that our learner $\mathcal{A}$ must know how to handle repeated samples. We may have them anyway in S, but running on a bootstrap sample we are sure to have them. Some learning algorithms suffer when there are repeated samples - as they cause numerical problems (for example, linear and logistic regression), while for others it isn't a problem (for example, decision trees and k-NN).

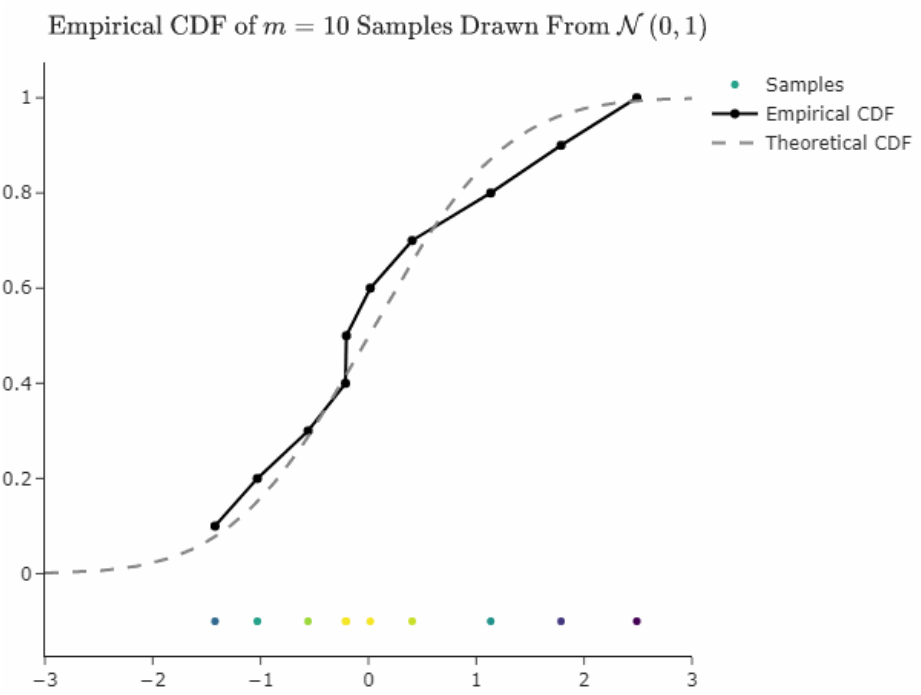


Figure 5.6: Empirical CDF: *of i.i.d samples drawn from a standard normal distribution. Chapter 5 Examples - Source Code*

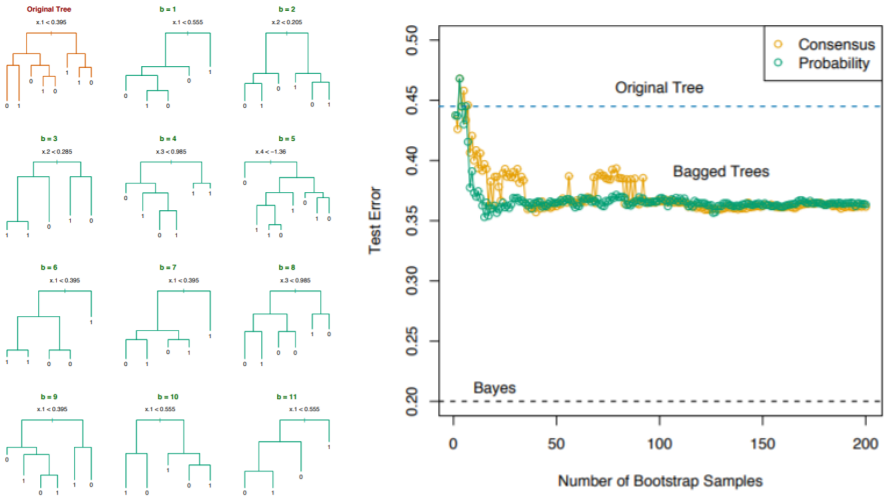


Figure 5.7: *Collection of Bagged Decision Trees. (Source: ESL)*

5.3.3 Bagging Reduces Variance

We saw that a committee majority vote reduces variance, but as seen in Figure 5.6 only to a certain degree. The amount of variance reduced is determined by the correlation between committee members. Therefore we can expect bagging to reduce variance as T increases, which will reduce the generalization error, but only proportionate to the correlation between the bagged prediction rules.

5.3.4 Random Forests - Bagging and De-correlating Decision Trees

As such we would like to find a way to de-correlate the committee members - namely, cause their predictions to somehow be less correlated. One way to do so is by slightly restricting each learner, in a random way, and hope that the performance gain (in bagging them) due to de-correlation is more than the performance loss to each learner by the restrictions. The most well know example of this principle is **Random Forests**. Ex.2

Recall the Decision Tree classification algorithm over $\mathcal{X} = \mathbb{R}^d$. We have a training sample S with m points. The Random Forest classifier is obtained by using Bagging on top of the Decision Tree algorithm, **with the important step** that drives the de-correlation: the algorithm has a tuning parameter $k \leq d$. When growing each decision tree, in each split, we choose uniformly at random a subset of k out of the d features. We choose the split only among these k features. Formally:

Algorithm 5 Random Forests

procedure RANDOM-FOREST(training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, maximal tree depth $R \in \mathbb{N}$, minimal training samples in any leaf m_{min}, number of trees T, number of coordinates to choose from in each split k)
  for $t = 1, \ldots, T$ **do**
    Draw a Bootstrap sample S^{*t} from S
    Train a decision tree $h_{S^{*t}}$ on the sample S^{*t} where at each split
    if Not reached maximal depth R or minimal number of samples m_{min} **then**
      Select uniformly at random k features from $[d]$
      Choose the best feature to split by from the st of k features
      Split based on selected feature and threshold
    end if
  end for
  return $h_S(\mathbf{x}) = sign\left(\sum_{i=1}^T w_t h_t(\mathbf{x})\right)$
end procedure

This de-correlation trick works: pretty much on every classification problem you will work on, you will observe something like the next plot: Bagging trees is much better than a single tree, and Random Forest (Bagging with the de-correlation trick) is better than just Bagging trees.

Bagging - Discussion Points

- **Can Bagging harm our prediction?** Always remember that a committee of fools (a committee where each member has probability $p < 0.5$ to make the right decision) makes worse decisions than a single member. So, when our base learner is so poor that its generalization loss is less than 0.5 we shouldn't use Bagging.
- **What are the disadvantages of Bagging?** As we train not one but T models we need to train all T of them. This directly increases time complexity by a factor proportionate to T. In addition, for predictions, as we need all T models we must store all T of them. Lastly, we loose interpretability as it is much harder to understand why the committee made the decision. We need to understand the decision of each of the T members.
- **Parallelizem** From the computational perspective, it is important to note that Bagging in general (and Random Forests in particular) is **embarrassingly parallelizable**. When training a Bagging model with T committee members, we can use T machines in parallel, each using

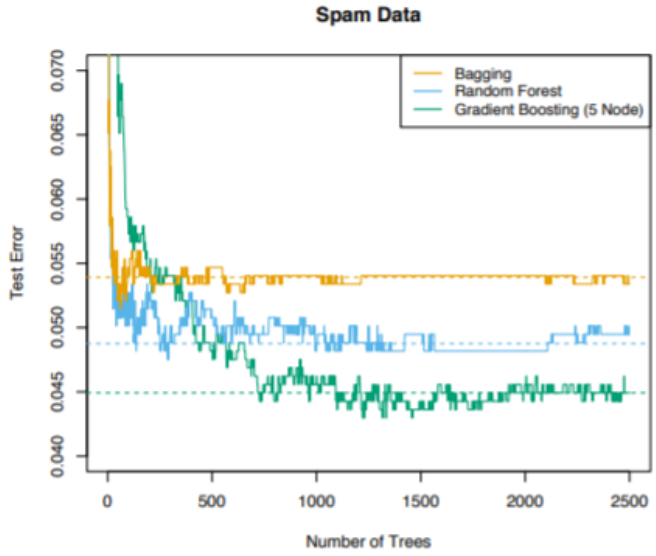


Figure 5.8: *Test error of simple Bagging of decision trees (no de-correlation), Random Forests, and Gradient Boosting of Trees. (Source: ESL)*

its own random seed to select Bootstrap samples (and random splits, in Random Forest). The machines do not need to interact; when each machine is done, it returns the committee member h_t to the master node.

- **Predicted Class Probabilities:** Can we use the **proportion** of the committee members who voted $+1$ as a predicted class probability? Estimated class probabilities are estimates of $\mathbb{P}\{Y = +1, | X = x\}$. The proportion of members who voted $+1$ estimates $\mathbb{P}\{h_S(x) = +1\}$, which is a different quantity.

5.4 Boosting

Bootstrap and Bagging produce an ensemble of classifiers each trained over a "new" artificial training sample generated from the original one. Boosting on the other hand utilizes the single training sample and produces different classifiers by assuming different distribution over these samples. In Boosting we take a "weak" learning algorithm, an algorithm with better-than-random but possibly not so good accuracy (i.e. generalization error) and *boost* it using a clever committee method to obtain a learning algorithm with good accuracy.

In Bagging, we *pretended* to have fresh training samples $S_1, \ldots, S_T$, and each committee member trained on a different sample. In Boosting on the other hand, we go even further and *pretend* to have *different underlying distributions* $\mathcal{D}$ from which the training sample is drawn. More specifically, in Boosting each committee member h_t is the result of running $\mathcal{A}$ against a training sample S_t that mimics an *i.i.d* sample of size m from a *different distribution* $\mathcal{D}^t$. Whereas in Bagging each committee member is trained independently of all other members, in Boosting the committee members are trained sequentially, one after the other, and each is an improvement, in some sense, on the previous one.

The clever idea behind Boosting is that after we finish training h_t, based on the distribution $\mathcal{D}^t$, we update the distribution in a way that increases the measure of samples where h_t was wrong. This way, we force h_{t+1} to try do better on those particular samples.

What is meant by "running $\mathcal{A}$ against the training sample S with distribution $\mathcal{D}^t$"? One way to interpret this is to take a **weighted Bootstrap** sample from S, where the probability of selecting $(x,y) \in S$ is proportional to $\mathcal{D}^t(x,y)$. A simpler way to interpret this is as follows. If $\mathcal{A}$ uses the ERM principle, say for standard misclassification ($0-1$ loss), namely, looking to minimize the empirical risk,

$$L_S(h) = \sum_{i=1}^{m} \mathbb{1}\left[y_i \neq h(\mathbf{x}_i)\right]$$

then we can use S itself and have the base learner minimize the **weighted** empirical risk

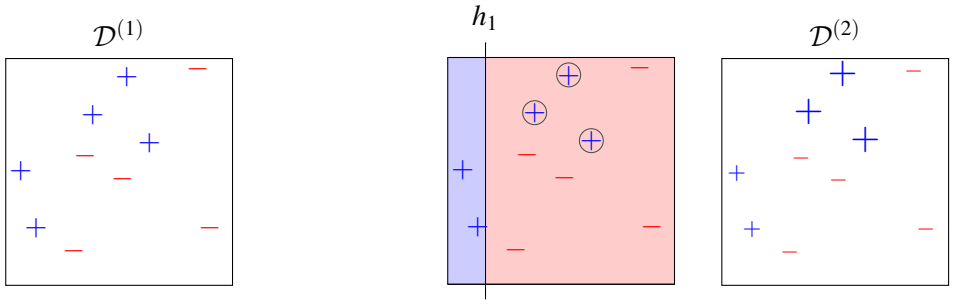
$$L_{S,\mathcal{D}^t}(h) = \sum_{i=1}^{m} \mathcal{D}_i^t \mathbb{1}\left[y_i \neq h(\mathbf{x}_i)\right]$$

where for each $(\mathbf{x}_i, y_i) \in S$ we write $\mathcal{D}_i^t := \mathcal{D}^t(\mathbf{x}_i, y_i)$, so that $\sum_{i=1}^m \mathcal{D}_i^t = 1$.

Observe that these two interpretations are equivalent in expectation. Indeed, the expected number of times for a sample $(\mathbf{x}_i, y_i)$ to appear in the weighted Bootstrap sample is $\mathcal{D}_i^t$, and so it would (in expectation) appear $\mathcal{D}_i^t$ times in the empirical risk sum.

Note that we usually prefer second option (using weighted empirical risk) to the first option (using weighted bootstrap). It's more computationally efficient, and does not require worrying about repeated samples. However, the first option (using weighted bootstrap) is always available. The second option (using weighted empirical risk) is not always possible, and is implemented ad-hoc for the particular base learner we are boosting.

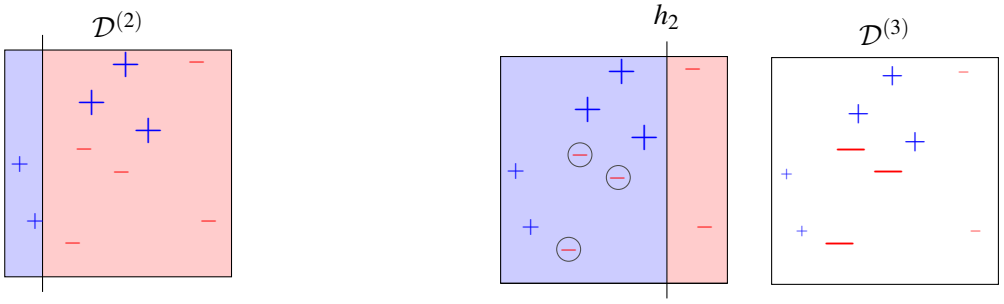
To understand this idea consider the following scenario. Suppose we begin with a training sample S of five positive and five negative samples as seen in Figure 5.9. At first we define a uniform distribution over the samples. Then suppose we fit a threshold classifier over S and $\mathcal{D}^{(1)}$ producing h_1. Notice that h_1 misclassified 3 samples. As such we update the distribution of weights over the samples increasing the weights of the misclassified samples and decreaseing the weights of the correctly classified samples, forming $\mathcal{D}^{(2)}$.
Next, we use $\mathcal{D}^{(2)}$ as the assumed sample distribution over S. By fitting a new threshold function, this time using $\mathcal{D}^{(2)}$ over S and minimizing the weighted misclassification error, we produce h_2. Similar to before based on the results of h_2 we form $\mathcal{D}^{(3)}$ where we increase the weights of the misclassified samples and decrease the weights of the correctly classified samples. This operation is done with respect to $\mathcal{D}^{(2)}$ (and not $\mathcal{D}^{(1)}$) as this is the distribution over which h_2 was produced. Notice that samples where both h_1 and h_2 have correctly classified are now with very low weights in $\mathcal{D}^{(3)}$.
Performing a third iteration the learner this time outputs h_3 which is able to correctly classify the samples misclassified in the previous iteration. Then we update the sample weights distribution based on the classification of h_3. Focusing for example on the bottom left negative sample, notice that all three learners have correctly classified this sample. Therefore, its weight under $\mathcal{D}^{(4)}$ is very low.



(a) *Initial uniform distribution over training sample*

(b) *Fitting h_1 over $\mathcal{D}^{(1)}$ and producing $\mathcal{D}^{(2)}$*

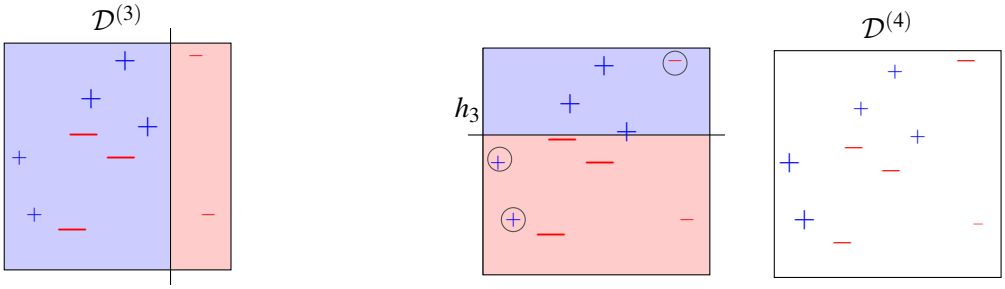
Figure 5.9: Boosting illustration: *Initial sample distribution and first iteration*



(a) *Current distribution over training sample*

(b) *Fitting h_2 over $\mathcal{D}^{(2)}$ and producing $\mathcal{D}^{(3)}$*

Figure 5.10: Boosting illustration: *second iteration over results of first iteration*



(a) *Current distribution over training sample*

(b) *Fitting h_3 over $\mathcal{D}^{(3)}$ and producing $\mathcal{D}^{(4)}$*

Figure 5.11: Boosting illustration: *third iteration over results of second iteration*

Then, if we used the three classifiers produced above h_1, h_2, h_3 and predict based on all of them we achieve a classifier with decision boundaries as in Figure **??**. Even though each single classifier has a simple decision boundary of a threshold function, the ensemble is able to describe much more

complex data scenarios. Classifying based on all three classifiers is done by

$$h_{boost}(\mathbf{x}) \coloneqq sign\left(\sum w_i h_i(\mathbf{x})\right)$$

where w_i are weights given to each classifier and reflect how successful that classifier is.

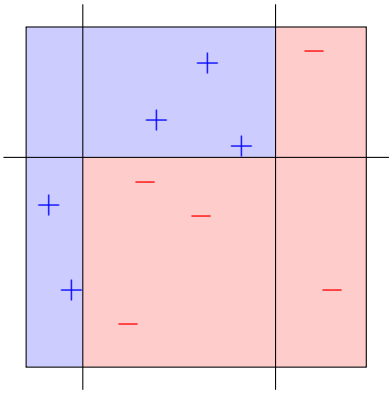


Figure 5.12: Boosting illustraion: *Decision boundaries of the ensemble of classifiers* h_1, h_2, h_3.

5.4.1 AdaBoost Algorithm

The original boosting meta-algorithm is known as **Ada**ptive **Boost**ing. The illustration above follows the operations done by the AdaBoost algorithm.

Algorithm 6 Adaptive **Boost**ing

procedure ADABOOST(training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, base learner $\mathcal{A}$, number of rounds T)
  Set initial distribution to be uniform: $\mathcal{D}^{(1)} \leftarrow \left(\frac{1}{m}, \ldots, \frac{1}{m}\right)$ ▷ Initialize parameters
  for $t = 1, \ldots, T$ **do**
    Invoke base learner $h_t = \mathcal{A}\left(\mathcal{D}^{(t)}, S\right)$
    Compute $\varepsilon_t = \sum \mathcal{D}^{(t)} \mathbb{1}\left[y_i \neq h_t(\mathbf{x}_i)\right]$
    Set $w_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) = \frac{1}{2}\ln\left(\frac{1}{\varepsilon_t} - 1\right)$.
    Update sample weights $\mathcal{D}_i^{(t+1)} = \mathcal{D}_i^{(t)} \exp\left(-y_i \cdot w_t h_t(\mathbf{x}_i)\right), \quad i = 1, \ldots, m$
    Normalize weights $\mathcal{D}_i^{(t+1)} = \frac{\mathcal{D}_i^{(t+1)}}{\sum_j \mathcal{D}_j^{(t+1)}} \quad i = 1, \ldots, m$
  end for
  return $h_S(\mathbf{x}) = sign\left(\sum_{t=1}^T w_t h_t(\mathbf{x})\right)$
end procedure

The idea is simple: from iteration t to iteration $t+1$, we want to **increase** the weights of samples misclassified by h_t (where $y_i h_t(\mathbf{x}_i) = -1$) and **decrease** the weights of samples correctly classified by h_t. We want to make the classification problem "maximally hard" in the sense that weighted empirical risk of h_t, with respect to the updated weights $\mathcal{D}^{t+1}$, is the worse possible, namely $1/2$. Finally, the prediction rules vote in the committee with weights w_t.

Claim 5.4.1 For the weighting factor of $w_t = \frac{1}{2} ln\left(\varepsilon_t^{-1} - 1\right)$ and $\varepsilon = \sum_{i=1}^{m} \mathcal{D}_i^t \cdot \mathbb{1}\left[y_i \neq h_t\left(\mathbf{x}_i\right)\right]$ the weighted empirical risk of h_t with respect to $\mathcal{D}^{t+1}$ is $1/2$:

$$\sum_{i=1}^{m} \mathcal{D}_i^{t+1} \cdot \mathbb{1}\left[y_i \neq h_t\left(\mathbf{x}_i\right)\right] = \frac{1}{2}$$

Proof. Directly expressing the weighted empirical risk:

$$\begin{aligned}
\sum_{i=1}^{m} \mathcal{D}_i^{t+1} \cdot \mathbb{1}\left[y_i \neq h_i\left(\mathbf{x}_i\right)\right] &= \frac{\sum_{i=1}^{m} \mathcal{D}_i^t \exp(-\mathbf{w}_t y_i h_t(\mathbf{x}_i) \mathbb{1}\left[y_i \neq h_t(\mathbf{x}_i)\right])}{\sum_{j=1}^{m} \mathcal{D}_j^t \exp\left(-\mathbf{w}_t y_j h_t\left(\mathbf{x}_j\right)\right)} \\
&= \frac{\exp(\mathbf{w}_t \varepsilon_t)}{\exp(\mathbf{w}_t \varepsilon_t) + \exp(-\mathbf{w}_t)(1-\varepsilon_t)} \\
&= \frac{\varepsilon_t}{\varepsilon_t + \exp(-2\mathbf{w}_t)(1-\varepsilon_t)} \\
&= \frac{\varepsilon_t}{\varepsilon_t + \frac{\varepsilon_t}{1-\varepsilon_t}(1-\varepsilon_t)} = \frac{1}{2}
\end{aligned}$$

■

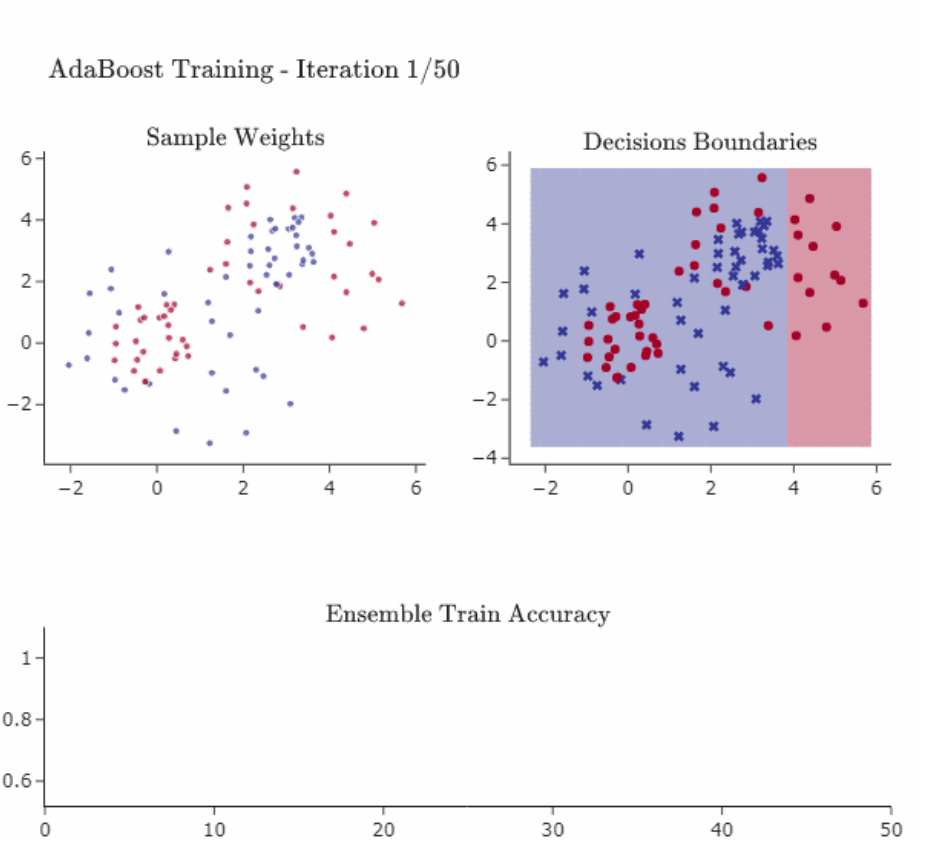


Figure 5.13: Adaboost fitting animation: *Chapter 5 Examples - Source Code*

5.4.2 PAC View of Boosting - Weak Learnability

Historically, Boosting appeared as an answer to a fascinating question for which we first need to define **Weak Learnability**:

Definition 5.4.1 — γ-Weak-Learner. A learning algorithm $\mathcal{A}$ is a γ-weak-learner for an hypothesis class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that

- For every $\delta \in (0,1)$
- For every distribution $\mathcal{D}$ over the sample space $\mathcal{X}$
- For every labeling function $f : \mathcal{X} \to \{\pm\}$

if the realizabiity assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running $\mathcal{A}$ on a training sample of $m \geq m_{\mathcal{H}}(0,1)$ *i.i.d* samples drawn according to $\mathcal{D}$ and labeled by f, the algorithm returns an hypothesis $h_S = \mathcal{A}(S)$ such that with probability at least $1-\delta$ (with respect to choice of the training sample S), we have $L_{\mathcal{D},f}(h_S) \leq 1/2 - \gamma$.

Definition 5.4.2 An hypothesis class $\mathcal{H}$ is γ-weak-learnable if there exits a γ-weak-learner for $\mathcal{H}$.

How is this different than PAC-learnability? If an hypothesis class $\mathcal{H}$ is PAC-learnable, then for **every** (ε, δ) there exists a learner $\mathcal{A}$. This means that we can learn and generalize a labeling function from $\mathcal{H}$ to any accuracy ε we want. But if $\mathcal{H}$ is γ-weak-learnable, for any δ **and just for** $\varepsilon = 1/2 - \gamma$ there is a learner $\mathcal{A}$. We may not be able to find a learner that has better accuracy (lower ε).

The question that motivated Boosting was the following:

- Suppose that $\mathcal{H}$ is PAC-learnable. Then we know that the rule $ERM_{\mathcal{H}}$ will learn (namely, will be probably approximately correct etc) with a near-minimal number of samples.
- But what if $ERM_{\mathcal{H}}$ is computationally hard? (we've seen examples)
- Assume we can find a **simple** hypothesis class (a "base hypothesis class") $\mathcal{H}_{base}$, such that $ERM_{\mathcal{H}_{base}}$ (choosing the hypothesis in $\mathcal{H}_{base}$ with lowest empirical risk) is computationally efficient, and is γ-weak-learner for $\mathcal{H}$ for some γ.
- This means that we have a computationally efficient way to learn with accuracy $1/2 - \gamma$, for some γ. Maybe we can't find an efficient learner with better γ.
- Is there a way to **boost** $ERM_{\mathcal{H}_{base}}$ in a computationally efficient way, and create a computationally efficient learner $\mathcal{A}$ which is close to minimizing ERM over $\mathcal{H}$?

For example, think about Decision trees. We saw that the ERM learner is not computationally feasible on this hypothesis class. But a small tree may be able to achieve accuracy (over a sample labeled by a larger tree) which is not great, but better than random. Well, as the following theorem shows, Adaboost does just that (for the full proof refer to UML 10.2)

Theorem 5.4.2 Let S be a training set. Assume that at each iteration of Adaboost, the base learner returns a prediction rule (hypothesis h_t) for which the weighted empirical risk satisfies:

$$\sum_{i=1}^{m} \mathcal{D}_i^t \mathbb{1}\left[y_i \neq h(\mathbf{x}_i)\right] \leq \frac{1}{2} - \gamma$$

Then the (standard, non-weighted) empirical risk of the output prediction rule of Adaboost, h_{boost}, (the weighted committee vote) satisfies:

$$L_S(h_{boost}) \equiv \frac{1}{m}\sum_{i=1}^{m} \mathbb{1}\left[y_i \neq h_{boost}(\mathbf{x}_i)\right] \leq \exp\left(-2\gamma^2 T\right)$$

5.4.3 Bias-Variance in Boosting

The hope is, of course, that we are not overfitting, so that low empirical risk will imply low generalization loss. Suppose we run T iterations of Adaboost over a learner $\mathcal{A}_{base}$ that returns hypothesis from $\mathcal{H}_{base}$. We would like to know what is the effective hypothesis class and how large is it. Adaboost with T iterations will return a function from the hypothesis class

$$\mathcal{H}_T := \left\{ \mathbf{x} \mapsto \sum_{t=1}^{T} w_t h_t(\mathbf{x}) \,\middle|\, w_t \in [0,\infty),\ \sum_t w_t = 1,\ h_t \in \mathcal{H}_{base} \right\}$$

namely convex combinations of hypotheses from $\mathcal{H}_{base}$. Therefore $\mathcal{H}_T$ becomes larger as T grows. Fortunately it does not grow too fast with T. Suppose for example we have a canonical way to measure the "size" of $\mathcal{H}_T$. It is possible to show that under certain conditions $VCdim(\mathcal{H}_T)$ is roughly $T \cdot VCdim(\mathcal{H}_{base})$. So we can expect Boosting to increase the variance (compared with the base learner) as T increases, but "not too fast".

On the other hand, it is clear that Boosting decreases bias. This is seen from the fact that the empirical risk decreases as T grows, meaning that $\mathcal{H}_T$ is able to come closer and closer to the labeling function on the training set. The fact that empirical risk decreases **exponentially** with T tells us that the bias decreases quite fast. Overall, Boosting typically decreases bias much faster than it increases variance, which is why it typically improves generalization loss quite dramatically. But the question that remains is if we use T too large, will boosting overfit?

Very often we see that boosting ERM over a very simple base hypothesis class is better than boosting ERM over a more complicated class. Figure 5.14 shows the example of the test error of boosting decision stumps (i.e. decision trees with a single split) with Adaboost over number of boosting iterations T, compared with the test errors of a single stump and of a single large decision tree. We are able to see that boosting this very simple base hypothesis class still achieves much better results compared to the very complex hypothesis class of trees with 244 nodes.

5.5 Summary and Exercises

In this chapter we discussed committee based decisions and saw that a committe of learners using majority vote will achieve better accuracy compared to a single member, given that each member decides better than a random guess. We also saw that this improvement increases as the size of the committee grows bu tthat is bounded by the correlation between the members. To apply this concept we introduced three general methods:

- *Bootstrap:* A method to generate "new" tranining samples from the one training sample we have.
- *Bagging:* A committee method where we run some base learner against bootstrap samples. Learners are unrelated to each other and all committee members have equal voting weight.

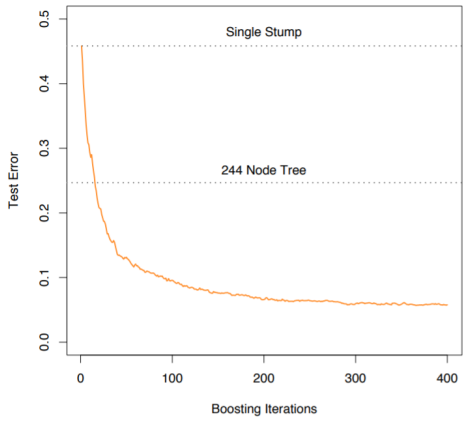


Figure 5.14: *Test error of boosting decision stumps with Adaboost over the number of boosting iterations*

- *Boosting:* A committee method where we run some base learner sequentially on weighted bootstrap samples. Sample weights change between iterations such that samples that the learner misclassified will increase in weight for the next iteration. The committee decision is based on the weighted vote of the members with weights related to their empirical loss.

These methods implement three general principles:

- We can create "artificial" training sets from our one training set S by sampling from S with replacements. This method is known as The Bootstrap. In a typical Bootstrap sample, about a third of the points a left out and others appear more than once.
- We can create a learner with improved accuracy and reduced variance by averaging better than random base learners. when the base learner gets a Bootstrap sample it is called Bagging. Bagging can be done in parallel as each prediction rule is created independently of the others. The prediction accuracy of the Bagging learner improves if the different prediction rules used are as de-correlated as possible. For example Random Forest achieves de-correlation by restricting each split in each tree to a random subset of coordinates.
- We can create a learner with with improved accuracy by boosting a base learner. The key idea behind Boosting is working with a probability distribution over S. Boosting means creating a weighted committee of prediction rules. Rules are created sequentially (not in parallel). Each rule is created the previous rule by modifying the distribution in such a way that misclassified training samples get an increased weight. For example Adaboost is a Boosting method that uses exponential updates to the probability distribution on S, such that the weighted empirical risk of the previous rule according to the updated distribution is exactly $1/2$ - the worse it can be.

Exercises

1. Consider the concept of boosting where we run $\mathcal{A}$ against a training sample with some distribution $\mathcal{D}$. Describe a possible implementation of a decision tree for each of the two interpretations:
    - Using weighted empirical risk: How would you change the CART algorithm to work with a given weight vector $\mathcal{D}^t$ over the training sample S? Hint: what is the best splitting

	Bagging	**Boosting**
Learns committee members	In parallel	Sequentially
Datasets for each member	Bootstrap samples	Weighted bootstrap or original sample with weighted ERM
De-correlation	Recommented	Not necessary
If T too large	Does not overfit	May overfit
Reduces	Variance	Bias
Committee vote is done	Unweighted	Weighted
Parallel computation implementation	Yes	No
With decision trees use	Deep trees	Shallow trees

Table 5.1: *Comparison of Bagging and Boosting*

now that we have weights?

- Using weighted Bootstrap: How would you change the CART algorithm to work with a given weight vector without changing the splitting algorithm, namely, by giving the algorithm a different training sample selected by weighted Bootstrap? Will the algorithm work with repeated samples?

2. Fill out a "Learner ID Card" for the Random Forest classifier specifying the following: hypothesis class; learning principle; computational implementation; making predictions, interpretability; providence of class probabilities; family of models; time complexity of training and predicting; storing the model.

6. Regularization and Model- Selection & Evaluation

6.1 Regularization

We have discussed multiple hypothesis classes and learners for choosing an hypothesis $h_s \in \mathcal{H}$ from a given hypothesis class $\mathcal{H}$. In most of these cases choosing h_S was based on minimizing some cost function $\mathcal{F}_S(h)$ over $h \in \mathcal{H}$. This means that $h_S := \mathcal{A}_0(S)$ is given by:

$$h_S := \operatorname*{argmin}_{h \in \mathcal{H}} \mathcal{F}_S(h)$$

The function $\mathcal{F}$ measures how well h fits the training sample S. We can call $\mathcal{F}_S(h)$ the **fidelity term**. We have seen a few examples for such learners already:

- In linear regression, $\mathcal{F}_S(h)$ measures the **RSS**.
- In logistic regression, $-\mathcal{F}_S(h)$ is the **likelihood** of the model (which we want to maximize).
- In SVM, $-\mathcal{F}_S(h)$ is the **margin** of the hyperplane (which we want to maximize).

With the simplest example for learners that minimize a cost function are of course the **ERM** learners. In any ERM based learner, we define some loss function $\ell(\cdot,\cdot)$ and define the empirical risk induced by ℓ, with respect to the training sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ to be:

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i)$$

For all of these the fidelity term $\mathcal{F}_S(h)$ is the empirical risk $L_S(h)$.

For the different hypothesis classes we have also discussed how their richness and expressiveness governs the bias-variance tradeoff. If the hypothesis class $\mathcal{H}$ is "too large", we are concerned that minimizing $\mathcal{F}$ over $\mathcal{H}$ may lead to over-fitting. Namely, we are concerned our learner will output

an hypothesis h_S which will not generalize well, as it is too well adapted to the particular training sample S.

One way to solve this problem would be to restrict $\mathcal{H}$. However, in such a case we are deliberately harming our performance. Instead, we would like to keep $\mathcal{H}$ large, but find a way to tell our learner $\mathcal{A}_0$ "Prefer simpler hypotheses but if you find a complex one that is *really* worth it - take it". To achieve this, we change the optimization problem that $\mathcal{A}_0$ uses to choose h_S. We introduce an additional term to the problem. We choose $\lambda \geq 0$ and define the learner $\mathcal{A}_\lambda : S \mapsto \mathcal{H}$ by:

$$h_S := \underset{h \in \mathcal{H}}{argmin}\, \mathcal{F}_S(h) + \lambda \mathcal{R}(h)$$

The term $\mathcal{R}$ is called the **regularization term**. If choosing $\mathcal{R}$ wisely then $\mathcal{R}(h)$ will measure the "complexity" of the hypothesis h. The more complicated the hypothesis h, the larger $\mathcal{R}(h)$ will be. So we see that in minimizing $\mathcal{F}_S(h) + \lambda \mathcal{R}(h)$ we now have a **trade-off**:

- On one hand, more complicated h, the better it can describe the training sample S, so the fidelity term $\mathcal{F}_S(h)$ will be smaller.
- On the other hand, the more complicated h, the larger $\mathcal{R}(h)$ will be.

And conversely, the simpler h, the larger the fidelity term $\mathcal{F}_S(h)$ (as it won't be able to describe the training sample very well) but the smaller $\mathcal{R}(h)$ will be. So λ is the parameter that governs the trade-off:

- For $\lambda = 0$, we have no regularization and are back to finding a minimizer of $\mathcal{F}_S(h)$.
- For $\lambda \to \infty$, the minimization problem pays no attention to the fidelity term and just wants to find the simplest possible $h \in \mathcal{H}$.
- Any finite value $\lambda \in (0, \infty)$ defines a specific trade-off between the need for fidelity (small $\mathcal{F}_S(h)$) and the need for simplicity of h (small $\mathcal{R}(h)$).

So, when we add regularization to the learner $\mathcal{A}_0 : S \mapsto \text{argmin}_{h \in \mathcal{H}} \mathcal{F}_S(h)$, we won't find a minimizer of $\mathcal{F}_S(h)$ over $\mathcal{H}$. We are hoping that a minimizer of the regularized objective function $\mathcal{F}_S(h) + \lambda \mathcal{R}(h)$ (which does not minimize $\mathcal{F}_S(h)$) will generalize better. This is because, the larger the value of λ, the simpler this minimizer will be.

We therefore get a **family** of learners $\{\mathcal{A}_\lambda\}_{\lambda \in [0,\infty)}$. The regularization parameter λ controls the bias-variance tradeoff: for $\lambda = 0$ we get the most variance and least bias (most complicated h); for $\lambda \to \infty$ we get the least variance and most bias (simplest h).

R We already saw one example for regularization - Soft SVM (subsection 3.3.3). Recall that the Soft-SVM classifier classifies using the half-space $\mathbf{w}^\perp$ where $\mathbf{w}$ is a solution to the optimization problem:

$$\begin{aligned} \text{minimize} \quad & \lambda ||\mathbf{w}||^2 + \frac{1}{m} \sum_i \xi_i \\ \text{subject to} \quad & y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i \ \wedge \ \xi_i \geq 0 \end{aligned}$$

Here, the fidelity term is $||\mathbf{w}||^2$. The smaller $||\mathbf{w}||^2$, the better we fit the training data (in the sense of larger margin). As the total margin is proportional to $1/||\mathbf{w}||$, so that minimizing $||\mathbf{w}||^2$ means maximizing the margin. The regularization term is $\frac{1}{m}\sum_i \xi_i$. The smaller this term

is, the "simpler" the hypothesis since we allow less violations of the margin. Note that here λ is placed on the fidelity term and not on the regularization term.

6.1.1 Regularized Decision Trees

When we presented the decision trees hypothesis class (section 3.6) and the CART algorithm for growing a classification tree we saw how to construct a tree of depth at most k following the ERM learning principle. As we have seen (Figure 3.15) as we increase k and enable deeper trees, though reducing the training error, we unfortunately overfit the training data and increase the generalization error. To avoid overfitting let us apply the concept of regularization to decision trees in the form of pruning: once a tree has been constructed, we remove some of the branches resulting in a less complex decision tree.

Regression Trees

Before introducing pruning as done in CART let us first understand how to adapt the algorithm for *regression trees*. Each classification tree is based on a tree partition of $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$ into N axis-parallel boxes where each tree $h \in \mathcal{H}_{CT}$ is a function $h : \mathbb{R}^d \to [C]$ defined by:

$$h(\mathbf{x}) = \sum_{j=1}^{N} c_j \mathbb{1}[\mathbf{x} \in B_j]$$

where $c_j \in [C]$ is the label associated with box j. In the case of regression trees c_j takes values in $\mathbb{R}$ instead of $[C]$.

Just as in the case of classification trees, given a training sample S, we would ideally want to search the hypothesis class of regression trees $\mathcal{H}_{RT}$ for a tree minimizing the empirical risk. As this is computationally hard we must use a heuristic to choose the tree based on the training sample.

Suppose we were given with some tree partition $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$, then the empirical risk with respect to the squared loss would be minimized by choosing the average response value of training samples in the box: Ex.1

$$c_j := \operatorname*{argmin}_{c \in \mathbb{R}} \sum_{i \,|\, \mathbf{x}_i \in B_j} (y_i - c)^2 = \frac{1}{|B_j|} \sum_{i \,|\, \mathbf{x}_i \in B_j} y_i$$

Then, we adjust the CART heuristic (Algorithm 3) for growing regression trees by replacing the misclassification loss by the square loss and the block response assignment to be:

$$\hat{y}_S(B) := \frac{1}{|B|} \sum_{i \,|\, \mathbf{x}_i \in B} y_i$$

Pruning a CART Regression Tree

The CART algorithm for growing a regression tree results in a mostly balanced tree. When growing a tree, with each split we do decreases the bias (as we have a more complicated hypothesis, with more expressive power) but also increases the variance (as the labels for prediction in the leaf are averaged over fewer training samples).

The last stage of CART (for both regression- and classification trees) is pruning the grown tree. This means decreasing the tree size by merging together some boxes that were split during the

growing stage. By doing so we aim to reach a better bias-variance tradeoff that will lead to better generalization.

Consider a training sample S and regression tree T induced by a partition $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$. The empirical risk of T on S is given by:

$$L_S(T) = \sum_{j=1}^{N} \sum_{i \,|\, \mathbf{x}_i \in B_j} (y_i - \hat{y}_S(B_j))^2$$

This will be the fidelity term $\mathcal{F}_S(T)$. For the regularization term we define $\mathcal{R}(T)$ to simply be $|T|$ the number of leaves (boxes) in T. In addition, for T_0 a fully grown tree created using the CART heuristic, denote $T \subset T_0$ if the tree T is a sub-tree of T_0 and can be obtained from T_0 by merging boxes in T_0. Then, the regularized optimization problem is given by

$$T^* = \operatorname*{argmin}_{T \subset T_0} L_S(T) + \lambda |T| \tag{6.1}$$

Namely, among all possible sub-trees of T_0 we look for the one optimally balancing empirical risk and hypothesis complexity. Note that we **do not** optimize this objective (6.1) over the entire hypothesis class $\mathcal{H}_{RT}^k$ which would be infeasible. We only consider sub-trees of the tree obtained by CART's greedy splitting, for which there exists efficient algorithms.

The last detail to complete the Random Forest model (subsection 5.3.4) is to incorporate the above pruning strategy into the constructed trees. Each single tree is grown using a heuristic (such as CART) that consists of two stages - growing a full tree and then pruning it. Therefore, we require three tuning parameters: the maximum number of levels, the minimum number of training samples in a leaf (a box) and the regularization parameter λ. When we run bagging on top of the CART algorithm, with the de-correlation trick (restricting each split to a random subset of coordinates), we get a random forest - either for regression or classification. However, in a typical random forest implementation there is no pruning stage. It is simply too computationally expensive to solve the pruning problem (6.1) for each tree if fitting a forest of hundreds of trees or more.

6.1.2 Regularized Regression

6.1.2.1 Subset Selection

Let us revisit linear regression and the least squares estimator. We defined the hypothesis class of linear regression as (2.1):

$$\mathcal{H}_{reg} := \left\{ h(x_1, \ldots, x_d) = w_0 + \sum_{i=1}^{d} x_i w_i \;\middle|\; w_0, w_1, \ldots, w_d \in \mathbb{R} \right\}$$

Then, given a regression problem $\mathbf{X}, \mathbf{y}$, we select $h_S \in \mathcal{H}_{reg}$ by finding the least squares estimator, derived using either the ERM learning principle and the squared loss or the Maximum Likelihood learning principle when assuming Gaussian noise. When doing so we assumed that the training sample size m was not smaller than the number of features d, and even preferred that $m \gg d$, since of $m \sim d$ the variance of the least squares estimator can be large.

However, in modern learning problems based on d features, very often we are in a situation

where d can be quite large. Linear regression was invented when features were measured and recorded manually. In the last few decades it became very easy to collect features and record them automatically, and a typical regression problem can easily have $d \sim m$ or even $d \gg m$.

When $d \sim m$ or, worse, $d \gg m$, we will have correlated features. The coefficients fitted by linear regression will be poorly determined. For example, a large positive coefficient for some feature can be canceled out by a large negative coefficient for an almost-parallel feature. So the linear regression learner we saw should not be used for $m \sim d$ and cannot be used for $d > m$.

Therefore, we would like to devise a method where we keep only a subset of the features. Let $k \leq d$ be the desired number of features, then we could solve the following optimization problem:

$$\begin{array}{ll} \underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{\text{minimize}} & ||w_0 \mathbf{1} + \mathbf{X}\mathbf{w} - \mathbf{y}||^2 \\ \text{subject to} & ||\mathbf{w}||_0 \leq k \end{array} \tag{6.2}$$

where $||\mathbf{w}||_0 \coloneqq \sum_i \mathbb{1}[\mathbf{w}_i \neq 0]$. That is, find a coefficients vector $\mathbf{w}$ that minimizes the *RSS* under the restriction of using exactly k features. This is exactly what the Best-Subset Selection algorithm does (Algorithm 7). It iterates over all possible combinations of k features, fits for each a least squares model and returns the model (i.e. a vector $\mathbf{w}$) achieving the lowest loss.

Algorithm 7 Best-Subset Selection

1: **procedure** BEST-SUBSET-SELECTION(X, y, k)
2: **for** $S \subset [d]$ where $|S| = k$ **do**
3: Fit regression model $\hat{\mathbf{w}}^{(S)}$ and compute $RSS\left(\hat{\mathbf{w}}^{(S)}\right) = \left|\left|\mathbf{y} - \mathbf{X}\hat{\mathbf{w}}^{(S)}\right|\right|^2$
4: **end for**
5: **return** $S^*, \hat{\mathbf{w}}^{(S^*)}$ such that $\text{argmin}_{\hat{\mathbf{w}}^{(S)}}\ RSS\left(\hat{\mathbf{w}}^{(S)}\right)$.
6: **end procedure**

As k, the number of features in the model, gives some measure of the complexity of the hypothesis, we can think of the family of learners $\left\{\mathcal{A}_k^{best-subset} | 0 \leq k \leq d\right\}$. As we change k we move on the bias-variance trade-off. If we restrict k to be small we will find simpler models with higher bias and lower variance and if we restrict k to be large we will find more complex models with lower bias and higher variance. We would like to find a value of k where on the one hand it is large enough to result in a model with low bias (and therefore has more descriptive power) but on the other hand small enough so to avoid high variances.

Unfortunately, even for a single value of k, in order to find the best-subset solution we must go over all $\binom{d}{k}$ subsets of k features. This is an NP-Hard problem and as such cannot be solved efficiently. However, if we change (6.2) to formulate a convex optimization problem with some convex function restricting the complexity of the model in some manner, then perhaps we will be able to find good approximations to the best-subset solution. In order to do so we first change to a related but not equivalent problem that introduces a regularization term over the number of used features. Now we jointly minimize the loss achieved by the model as well as the number of features it effectively contains.

$$\hat{\mathbf{w}}_{\lambda}^{subset} := \underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \; ||w_0 \mathbf{1} + \mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \lambda \, ||\mathbf{w}||_0 \tag{6.3}$$

where $\lambda \geq 0$. Notice how in both optimization problems (6.2) and (6.3) we do not include the intercept (w_0) in the restriction on the number of features as the intercept is not one of the features per se.

6.1.2.2 Ridge (ℓ_2) Regularization

In addition to the computational challenges of the best-subset selection, it often suffers from a high variance as features are included or excluded based on the single specific training-set we have. To cope with both problems, we use different *shrinkage methods* where we restrict the values of the coefficients, shrinking them (often) towards zero. One such method is the Ridge regression, which imposes a $||\cdot||_2$ penalty on the coefficients:

$$\widehat{\mathbf{w}}_{\lambda}^{ridge} := \underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} \; ||w_0 \mathbf{1} + \mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \lambda \, ||\mathbf{w}||_2^2 \quad \lambda \geq 0 \tag{6.4}$$

The regularization parameter $\lambda \geq 0$ is the complexity parameter that controls the amount of shrinkage. For $\lambda = 0$ we get the least squares solution. As $\lambda \to \infty$ we penalize more and more on the size of the coefficients vector, driving the solution $\widehat{\mathbf{w}}_{\lambda}^{ridge} \to 0$. As λ increases the bias increases (we restrict ourselves to specific sets of solutions) and variance decreases (solution is not based solely on training-set) where finally as $\lambda \to \infty$ then $\widehat{\mathbf{w}}_{\lambda}^{ridge} \to 0$.

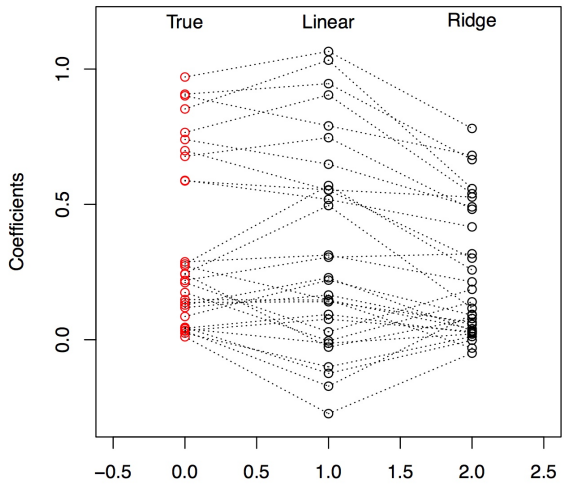


Figure 6.1: *Coefficient shrinkage induced by Ridge regression: Shows the change of the true coefficients of a linear model $y = \mathbf{x}^\top \mathbf{w} + \varepsilon$ between the fitted least squares solution (center) and the ridge solution (right). Source: Ryan Tibshirani Data Mining slides, CMU 36-462/36-662*

Notice that the ridge optimization problem is a convex optimization problem and can be solved using some generic QP solver. However, in the case of ridge regression there is no need as we can derive a closed form solution for the minimizer by applying the same strategy as we did for solving the ordinary least squares optimization problem.

Theorem 6.1.1 Let $\mathbf{X}, \mathbf{y}$ be a regression problem and $\lambda \geq 0$. Let $\mathbf{X} = U\Sigma V^\top$ be the SVD of $\mathbf{X}$. The Ridge estimator is given by:

$$\widehat{\mathbf{w}}_\lambda^{ridge} = U\Sigma_\lambda V^\top \mathbf{y}, \quad [\Sigma_\lambda]_{ii} = \frac{\sigma_i}{\sigma_i^2 + \lambda}$$

Proof. Let us replace $\mathbf{X}$ with its SVD:

$$\begin{aligned}
\widehat{\mathbf{w}}_\lambda^{ridge} &= \left(\mathbf{X}^\top\mathbf{X} + \lambda I\right)^{-1}\mathbf{X}^\top\mathbf{y} &&= \left(\left(U\Sigma V^\top\right)^\top U\Sigma V^\top + \lambda I\right)^{-1}\left(U\Sigma V^\top\right)^\top\mathbf{y} \\
&= \left(V\Sigma U^\top U\Sigma V^\top + \lambda I\right)^{-1} V\Sigma U^\top\mathbf{y} &&= \left(V\Sigma^2 V^\top + \lambda VV^\top\right)^{-1} V\Sigma U^\top\mathbf{y} \\
&= V\left(\Sigma^2 + \lambda I\right)^{-1} V^\top V\Sigma U^\top\mathbf{y} &&= V\Sigma_\lambda U^\top\mathbf{y}
\end{aligned}$$

where we denote $[\Sigma_\lambda]_{ii} = \frac{\sigma_i}{\sigma_i^2+\lambda}$ and σ_i are the singular values of $\mathbf{X}$. ■

If we choose a strictly positive λ then it can be shown that the Ridge solution takes the form of:

$$\widehat{\mathbf{w}}_\lambda^{ridge} := \left(\mathbf{X}^\top\mathbf{X} + \lambda I_d\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

where $\mathbf{X}^\top\mathbf{X} + \lambda I_d$ is a non-singular matrix for any $\lambda > 0$ regardless to $\mathbf{X}^\top\mathbf{X}$ being singular or non-singular. Unlike the least squares estimator which we have seen to be an unbiased estimator (2.2.1), the ridge estimator is biased. However, it reduces the variance enough such that the overall MSE is lower compared to the least squares solution. Ex.2 Ex.3

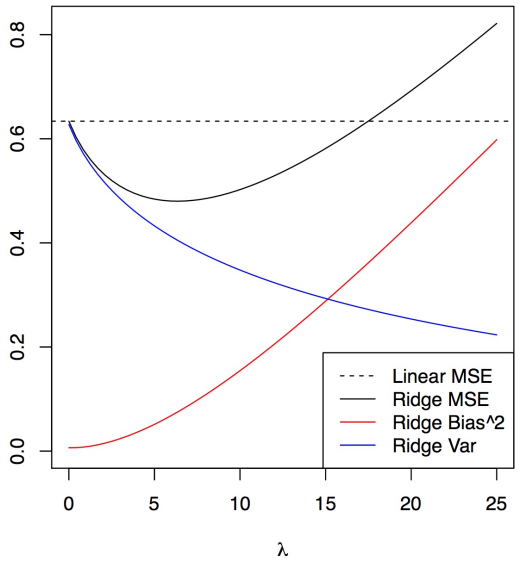


Figure 6.2: Bias, Variance and MSE of Ridge Solution: *as function of λ and compared to the least squares solution. Source: Ryan Tibshirani Data Mining slides, CMU 36-462/36-662*

Regularization Path

It is interesting to trace each individual weight $\hat{w}_i$ as λ changes. Figure 6.3 shows the *regularization path* of the ridge solution: it shows how the individual weights change for different values of lambda. Observe how the weights start from the linear regression weights ($\lambda = 0$) and shrink towards zero with a decay roughly like $1/\lambda$ as λ grows. Below the regularization path the losses achieved by the model fitted with each λ are shown separated to the loss of the fidelity term, the regularization term and the joint loss of the fidelity- and regularization terms together.

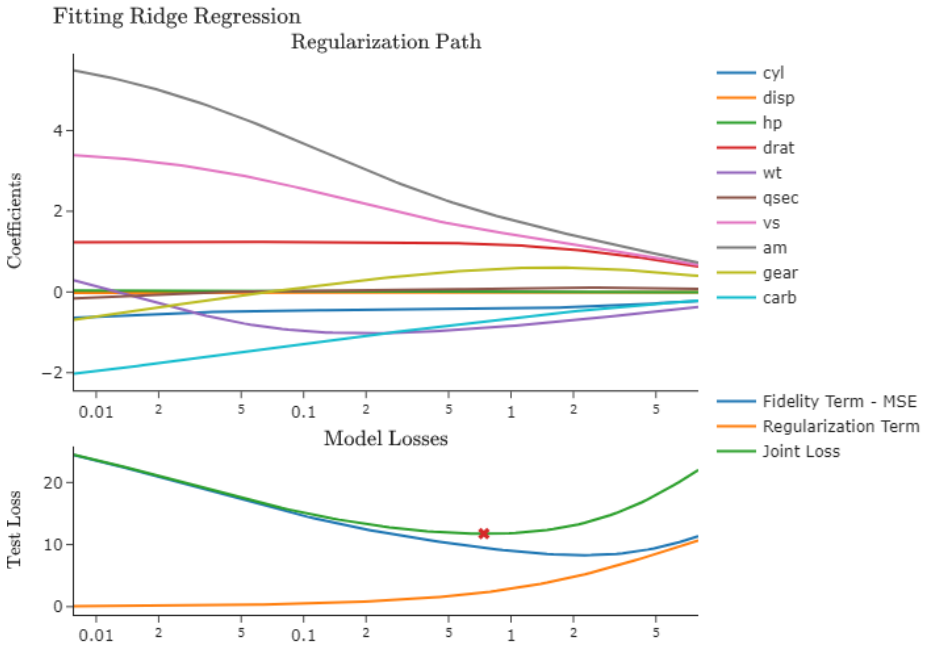


Figure 6.3: Ridge Regularization Path and Losses *as a function of λ over the Motor Trend Car Road Tests dataset. Chapter 6 Examples - Source Code*

6.1.2.3 Lasso (ℓ_1) Regularization

By introducing the ridge regularizer we were able to apply linear regression when $d > m$ or when $\mathbf{X}^\top\mathbf{X}$ is not invertible and that it provides a good way to get a simple handle on the bias-variance tradeoff of linear regression. In addition, we saw that even though the ridge estimator is a biased estimator it achieves a lower variance compared to the least squares estimator. However, as evident from the regularization path, the ridge estimator does not do what best-subset does. That is, it cannot **select** a specific subset of features to use for regression.

The *Least Absolute Shrinkage and Selection Operaotr* (Lasso) attempts to achieve exactly that. This regularization method uses the ℓ_1 norm rather than the ℓ_2 norm:

$$\widehat{\mathbf{w}}_\lambda^{lasso} := \underset{w_0\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d}{argmin} \; ||w_0\mathbf{1} + \mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \lambda\,||\mathbf{w}||_1 \quad \lambda \geq 0 \tag{6.5}$$

Similar to Ridge, when setting $\lambda = 0$ we get the least squares solution and setting $\lambda \to \infty$ will shrink the coefficients $\widehat{\mathbf{w}}_\lambda^{lasso} \to 0$. However the manner of shrinkage is very different between the two problems. The lasso solution is **sparse**. That is, it has zero entries in the vector. Therefore, the larger

λ, typically the more zeros $\widehat{\mathbf{w}}_\lambda^{lasso}$ will have, effectively defining a subset of features that are used by the model (a feature corresponding a coefficient of zero is not used by the model). We often refer to this subset of features as the "active set". As we are considering problems where d can be very large, it is hard to interpret the model. Thus, the Lasso has an important advantage in interpretability over Ridge, as it selects for us a subset of variables.

The Lasso optimization problem (6.5) is a convex problem and specifically is a quadratic program. There are however specialized solvers for the Lasso wihch calculate the entire regularization path at the same computational cost as solving Ridge.

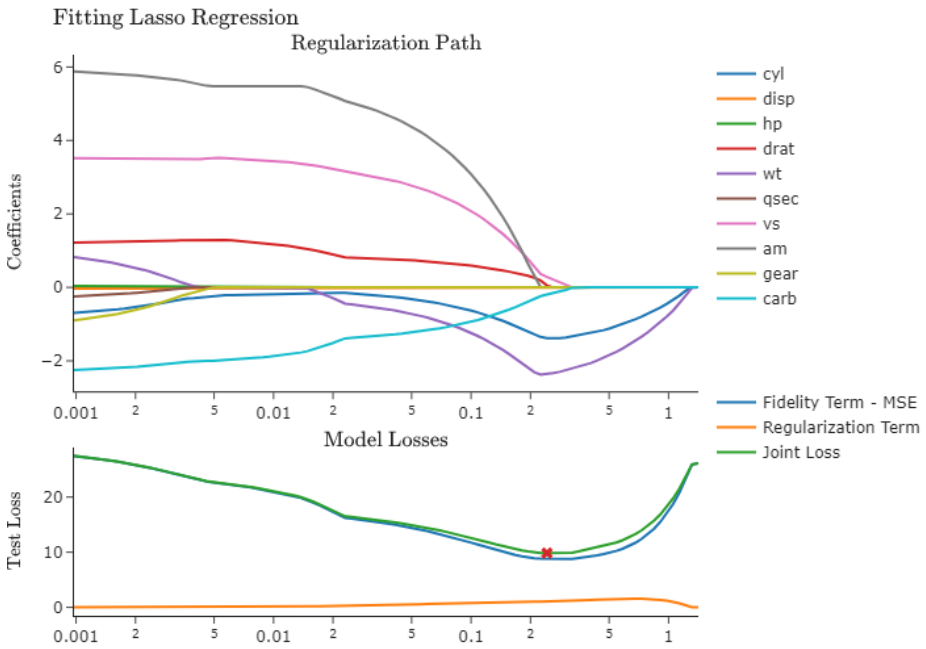


Figure 6.4: Lasso Regularization Path and Losses *as a function of λ over the Motor Trend Car Road Tests dataset. Chapter 6 Examples - Source Code*

Convexity vs. Sparsity

For the three regression regularization methods using the regularization terms of $||\cdot||_0, ||\cdot||_1, ||\cdot||_2$ (also denoted by ℓ_0, ℓ_1, ℓ_2), we saw that using ℓ_1, ℓ_2 we still have a convex optimization problem whereas for the ℓ_0 term the problem is non-convex. In addition, we stated that unlike ℓ_2, when using ℓ_1 we typically introduce sparsity to the solution. Let us expand this discussion for the L_q family of norms:

$$\text{For } 0 < q \in \mathbb{R}, x \in \mathbb{R}^d \quad ||x||_q := \left(\sum_{i=1}^{d} (x_i)^q \right)^{\frac{1}{q}} \tag{6.6}$$

Specifying any norm in this family as the regression regularization term then:

- Sparsity - For any $q \leq 1$ we typically obtain sparse solutions and therefore our active set of features is smaller than d.
- Convexity - For any $q \geq 1$ the optimization problem (when we use the RSS loss as the fidelity term) is convex. Therefore the problem can be solved efficiently.

The *Lasso* regression uses $q = 1$ and as such achieves **both** sparsity and convexity. But why are $L_{q\leq 1}$ norms obtain their sparsity? This is in fact based on the shapes of their unit balls.

Definition 6.1.1 For a norm $||\cdot||$ on $\mathbb{R}^d$, a ball of radius ρ (around the origin) is the set: $B_\rho := \{\mathbf{w} \in \mathbb{R}^d | ||\mathbf{w}|| \leq \rho\}$. The unit ball of a norm is the ball of radious $\rho = 1$.

For $L_{q>1}$, such as in the case of the Euclidean norm, the unit ball has no corners. In contrast, the unit balls of $L_{q\leq 1}$ have corners. Specifically, the unit ball of the ℓ_1 norm over $\mathbb{R}^d$ has $2d$ corners.

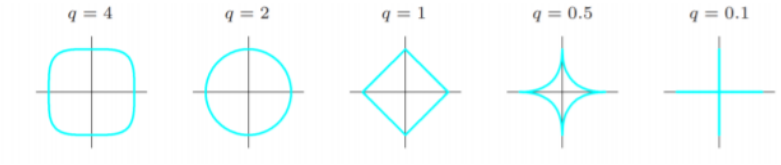


Figure 6.5: Unit Balls of Different L_q Norms: *Source: Elements of Statistical Learning, Figure 3.12*

Now, let us revisit the Ridge and Lasso optimization problems. Instead of the unconstrained forms seen in 6.5 and 6.4 consider the equivalent constraint versions:

minimize $w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d$	$\|\|w_0\mathbf{1} + \mathbf{X}\mathbf{w} - y\|\|^2$	minimize $\mathbf{w}_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d$	$\|\|w_0\mathbf{1} + \mathbf{X}\mathbf{w} - y\|\|^2$
subject to	$\|\|\mathbf{w}\|\|_2^2 \leq \rho$	subject to	$\|\|\mathbf{w}\|\|_1 \leq \rho$

If we fix some $\rho \in \mathbb{R}$ we can ask where in $\mathbb{R}^d$ will we find the $\widehat{\mathbf{w}}_\rho^{ridge}$ and $\widehat{\mathbf{w}}_t^{lasso}$ solutions. As the fidelity term is a quadratic form in $\mathbf{w}$, its level sets are ellipsoids. Suppose ρ is very large, so there is effectively no constrain, the solution of the minimization problem will be the least squares solution. As we restrict the solution more and more, by making ρ smaller and smaller, we are limiting the solution to be inside the norm ball. By definition, the solution will be found where one of the level sets of the fidelity term intersects with the ball of radius ρ. When we consider the $L_{q\leq 1}$ norms (such as the ℓ_1 in Lasso) this intersection typically happens at one of the corners of the norm ball. As these corners take place on the axes they correspond to **sparse** solutions.

6.1.2.4 The Orthogonal Design Case

Here is another way to understand why Lasso solutions are sparse. Consider the case where all features are orthogonal to each other, namely $\mathbf{X}^\top\mathbf{X} = I_d$. In some sense, this is the simplest setup for regression problems, where we can write $\mathbf{y}$ as a linear combination of **orthonormal** vectors. This is also known as an orthogonal design. In such setup, we have a closed form solution for $\widehat{\mathbf{w}}_\lambda^{subset}, \widehat{\mathbf{w}}_\lambda^{ridge}, \widehat{\mathbf{w}}_\lambda^{lasso}$ based on the $\widehat{\mathbf{w}}^{LS}$ solution.

Let us define two *thresholding* functions.

Definition 6.1.2 Let the hard- and soft-threshold (at λ) be the functions $\eta_\lambda^{hard}, \eta_\lambda^{soft} : \mathbb{R} \to \mathbb{R}$

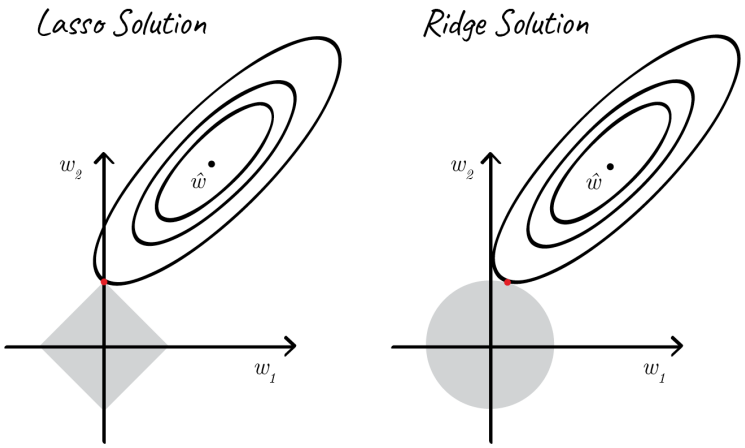


Figure 6.6: The constrained optimization problems in $\mathbb{R}^2$***:*** *With the Lasso problem with the* ℓ_1 *unit ball and Ridge problem with the* ℓ_2 *unit ball. The red dots represent the coefficients vector that is both on the unit ball and the objective function.*

defined by:

$$\eta_\lambda^{hard} := \mathbb{1}\left[|x| \geq \lambda\right] \cdot x \qquad \eta_\lambda^{soft} := sign\left(x\right)\left[|x| - \lambda\right]_+ = \begin{cases} x-\lambda & x \geq \lambda \\ 0 & |x| < \lambda \\ x+\lambda & x \leq \lambda \end{cases}$$

These functions define a manner of shrinking an input value x towards 0, depending on λ. The hard-thresholding zeros inputs in the range of $(-\lambda, +\lambda)$ and leaves inputs outside of this range unchanged. The soft-thresholding shrinks inputs out of the $(-\lambda, +\lambda)$ range, and zeros inputs that are within the range.

Claim 6.1.2 Let X, y be an orthogonal design matrix and a response vector. Denote $\widehat{w}$ the OLS solution. The lasso regularization optimization problem takes the form of $\widehat{w}^{lasso}\left(\lambda\right) = \eta_\lambda^{soft}\left(\widehat{w}\right)$

Proof. Recall that the least squares solution is $\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}\mathbf{y} = \mathbf{X}\mathbf{y}$. We can re-write the objective function as:

$$\begin{aligned} f_{\ell_1}\left(\mathbf{w}\right) = \tfrac{1}{2}\left|\left|\mathbf{y} - \mathbf{X}\mathbf{w}\right|\right|_2^2 + \lambda\left|\left|\mathbf{w}\right|\right|_1 &= \tfrac{1}{2}\left(\left|\left|\mathbf{y}\right|\right|^2 - 2\mathbf{y}^\top\mathbf{X}\mathbf{w} + \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}\right) + \lambda\left|\left|\mathbf{w}\right|\right|_1 \\ &= \tfrac{1}{2}\left(\left|\left|\mathbf{y}\right|\right|^2 + \left(\mathbf{w}^\top - 2\hat{\mathbf{w}}^\top\right)\mathbf{w}\right) + \lambda\left|\left|\mathbf{w}\right|\right|_1 \\ &= \tfrac{1}{2}\left|\left|\mathbf{y}\right|\right|^2 + \textstyle\sum_{j=1}^d\left(\tfrac{1}{2}w_j^2 - \hat{w}_j w_j + \lambda\left|w_j\right|\right) \\ &= \tfrac{1}{2}\left|\left|\mathbf{y}\right|\right|^2 + \textstyle\sum_{j=1}^d\left(\tfrac{1}{2}w_j - \hat{w}_j + \lambda sign\left(w_j\right)\right)w_j \end{aligned}$$

Let us minimize the expression for each w_j taking into account the value of λ. So:

$$\frac{\partial\frac{1}{2}\left|\left|\mathbf{y}-\mathbf{X}\mathbf{w}\right|\right|+\lambda\left|\left|\mathbf{w}\right|\right|_1}{\partial w_j} = \frac{\partial\left(\frac{1}{2}w_j - \hat{w}_j + \lambda sign\left(w_j\right)\right)w_j}{\partial w_j} = w_j - \hat{w}_j + \lambda sign\left(w_j\right) = 0$$

$$\Downarrow$$

$$w_j = \hat{w}_j - \lambda sign\left(w_j\right)$$

Let us divide into cases:

- If $|\hat{w}_j| < \lambda$ then $w_j = 0$. That it because:
  - If $w_j < 0$ then $0 > w_j = \hat{w}_j + \lambda$, which $\forall |\hat{w}_j| < \lambda$ the right-hand side is positive in contradiction.
  - If $w_j > 0$ then $0 < w_j = \hat{w}_j - \lambda$, which $\forall |\hat{w}_j| < \lambda$ the right-hand side is negative in contradiction.
- If $\hat{w}_j \geq \lambda$ then $w_j = \hat{w}_j - \lambda sign(w_j)$ which is valid only for $w_j \geq 0$ which means that $w_j = \hat{w}_j - \lambda$.
- If $\hat{w}_j \leq -\lambda$ then $w_j = \hat{w}_j - \lambda sign(w_j)$ which is valid only for $w_j \leq 0$ which means that $w_j = \hat{w}_j + \lambda$.

Putting it all together we got that:

$$w_j(\hat{w}_j, \lambda) = \begin{cases} \hat{\mathbf{w}} - \lambda & \hat{\mathbf{w}} \geq \lambda \\ 0 & |\hat{\mathbf{w}}| < \lambda \\ \hat{\mathbf{w}} + \lambda & \hat{\mathbf{w}} \leq \lambda \end{cases} \quad \Longrightarrow \quad \hat{w}_j^{lasso}(\lambda) = \eta_\lambda^{soft}(\hat{w}_j)$$

■

Similarly, we can obtain the Best-Subset and Ridge solutions:

$$\begin{aligned} \widehat{\mathbf{w}}_\lambda^{subset} &:= \eta_{\sqrt{\lambda}}^{hard}\left(\widehat{\mathbf{w}}^{LS}\right) \\ \widehat{\mathbf{w}}_\lambda^{ridge} &:= \widehat{\mathbf{w}}^{LS}/(1+\lambda) \end{aligned}$$

Namely, in the orthogonal design setup we can obtain the different solutions by applying a **univariate shrinkage function** to each coordinate of $\widehat{\mathbf{w}}^{LS}$ separately. Observe that in this case:

- The Best-Subset sets some coefficients to zero and leaves the rest untouched.
- The Lasso solution sets some coefficients to zero and shrinks the rest by λ.
- The Ridge solution simply multiplies by a scalar.

Therefore, for the Best-Subset and Lasso solutions, the solutions' sparsity grows as λ grows.

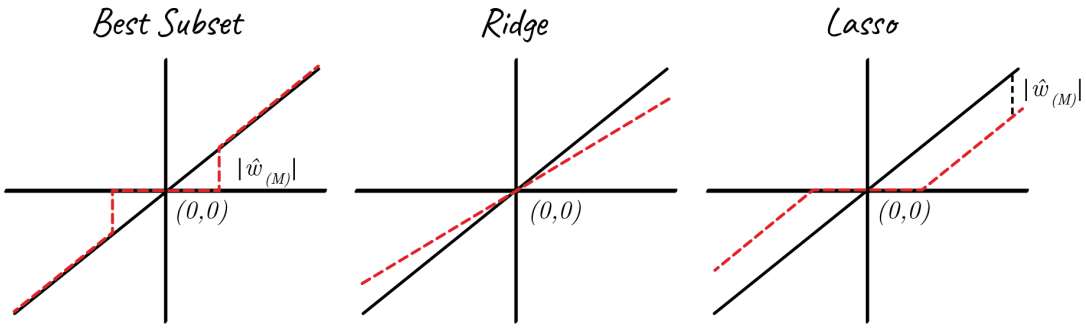


Figure 6.7: Shrinkage of Best-Subset, Ridge and Lasso in the orthogonal design case

6.1.3 Regularized Logistic Regression

We have seen a few different (linear) regression methods using regularization. We can also apply these regularization terms to other regression models such as the logistic regression. Similarly to

linear regression, if $m \sim d$ or $d > m$, also for logistic regression optimization will be numerically unstable, might have parallel feature vectors with large coefficients of opposite signs and hard for interpretation. All these problems are alleviated by adding a regularization term.

Recall that the logistic regression classifier find the coefficients vector by solving:

$$\hat{\mathbf{w}} := \underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{\operatorname{argmax}} \sum_{i=1}^{m} \left[y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + w_0 \right) - log \left(1 + e^{w_0 + \langle \mathbf{w}, \mathbf{x}_i \rangle} \right) \right]$$

We can add the ℓ_1 regularization term to the above fidelity term to obtain the ℓ_1-regularized logistic regression classifier:

$$\widehat{\mathbf{w}} := \underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{\operatorname{argmax}} \overbrace{\sum_{i=1}^{m} \left[y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + w_0 \right) - log \left(1 + e^{w_0 + \langle \mathbf{w}, \mathbf{x}_i \rangle} \right) \right]}^{\mathcal{F}_S(\mathbf{w})} + \lambda \overbrace{||\mathbf{w}||_1}^{\mathcal{R}(\mathbf{w})}$$

This is still a convex optimizatoin problem, and fast specialized solvers are available. The ℓ_1-regularized logistic regression classifier is a very powerful classifier over $\mathcal{X} = \mathbb{R}^d$. It has low variance, we are able to control the bias-variance tradeoff (by choosing λ) and is it very interpretable.

6.2 Model Selection and -Evaluation

So far, we have discussed several learning algorithms and meta-algorithms that can be applied over the existing learning algorithms. In the following we will be answering the following questions:

- How to **select** a model? For different algorithms we have described the existance of a family of learners where we defined some "tuning" hyper-parameter. For example, k the number of neighbors used in the k-NN algorithm; the maximal tree depth and pruning regularization in CART; or the regularization lambda in soft-SVM and the different regularized linear- and logistic- regression problems.

  In the case of meta-algorithms such as bagging or boosting, in addition to the base learners' tuning parameters, we need to choose the number of bagging/bootstrapping iterations B. If we are using de-correlation in bagging, there might be additional tuning parameters (e.g. the number k of allowed coordinated for a split in the Random Forest algorithm).

- How to **estimate** the performance of a chosen model? Before applying the chosen model on new samples we would like to estimate how will it perform on a new independent set of samples. If our estimation of the generalization error for the selected model is poor, perhaps we are working with the wrong learning algorithm for our problem, and should therefore select a different candidate. In addition, often we are interested in knowning how our chosen learner will perform before we begin using it.

The performance we would like to estimate, which would also influence the selection of the model, is the generalization error. Assume some loss function $\ell(\cdot, \cdot)$ then we defined the empirical risk of

an hypothesis $h : \mathcal{X} \to \mathcal{Y}$ simple as the average loss over the training sample:

$$L_S(h) := \frac{1}{m}\sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i) \quad S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$$

The generalization error (also reffered to as *test error*), that is the predicted error over an independent sample set depends on our data-generation model:

- No data-generation model: If we are not assuming any model to describe how the data is generated we simply define the generalization error as the average loss over the test set:

$$L(h) := \sum_{i=1}^{|T|} \ell(h(\mathbf{x}_i), y_i)$$

- Probability distribution over $\mathcal{X}$ and unknown labeling function $f : \mathcal{X} \to \mathcal{Y}$: When assuming a model as the PAC model, the generalization error is the expected error over sampling from the distribution:

$$\mathbb{E}_{x\sim\mathcal{D}}[\ell(h(\mathbf{x}), f(\mathbf{x}))]$$

- Probability distribution over $\mathcal{X} \times \mathcal{Y}$: When assuming a model as the Agnostic PAC model, the generalization error is the expected error over sampling a sample-label pair from the distribution:

$$L_{\mathcal{D}}(h) := \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h(\mathbf{x}), y)]$$

Once we understand what is correct generalization error, the next step is to find a way to **correctly estimate** it. Suppose we perform the following procedure: given a supervised batch learning setting, where our training sample is $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, let us perform model selection and -evaluation over a family of learning algorithms $\{\mathcal{A}_\alpha\}$ simply by using S:

Algorithm 8 Model Selection & Evaluation

1: **procedure** SELECT-AND-EVALUATE(S, $\{\mathcal{A}_\alpha\}$)
2: Train each model over S to obtain $\{h_\alpha := \mathcal{A}_\alpha(S)\}$.
3: Choose the best model by using $\alpha^* := \text{argmin}_\alpha L_S(h_\alpha)$
4: Estimate generalization error of h_{α^*} over train set $L_{\mathcal{D}}(h_{\alpha^*}) := L_S(h_{\alpha^*})$.
5: **return** $\alpha^*, h_{\alpha^*}, L_S(h_{\alpha^*})$
6: **end procedure**

This procedure will yield poor results (Figure 6.8). The selection is based on the empirical error of the model. As we have already seen in the bias-variance trade-off, the generalization error can be represented as the sum of both the bias- and the variance of the model. The model we select α^* will adapt as much as possible (under the restriction to some $\mathcal{H}$) to the training data. The more variance the learner has, the more it will be able to adapt to the particular properties of S. Our generalization error estimator will suffer from **optimism** - which is the technical term for the difference between the empirical risk and generalization error). Thus, we need to devise a different method to estimate the generalization error. One that will suffer less from optimism.

6.2.1 Train-Validation-Test Scheme

The naive approach of using the finite training set S both for training the model and estimating it's future performance yields poor results. Suppose we had infinitely many samples, could we have

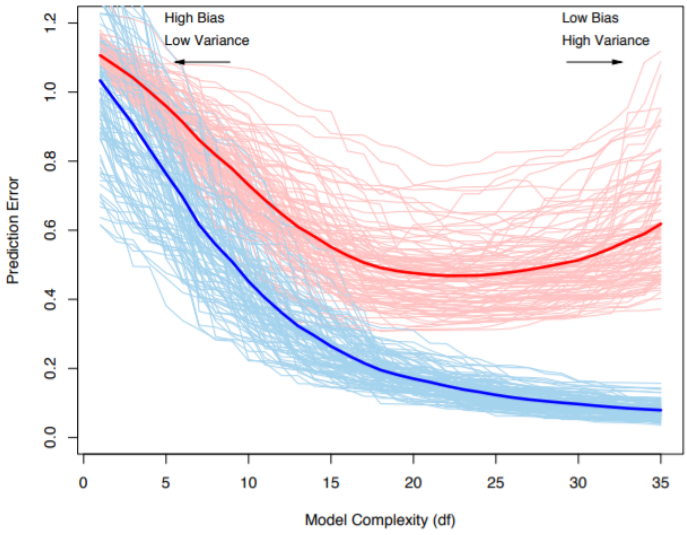


Figure 6.8: Train- and generalization Error: *as function of model complexity. For each pair of random train set and test set the blue line shows the train error as a function of model complexity while the red line shows the test error as a function of model complexity .Source: Elements of Statistical Learning*

done better? In such scenario we could use three different sets S (training), V (validation) and T (testing). Then we would perform model selection and evaluation as seen in Algorithm 9.

Algorithm 9 Model Selection & Evaluation

1: **procedure** SELECT-AND-EVALUATE(S, V, T $\{\mathcal{A}_\alpha\}$)
2: Train each model over S to obtain $\{h_\alpha := \mathcal{A}_\alpha(S)\}$.
3: Choose the model minimizing the loss over the *validation set*: $\alpha^* := \text{argmin}_\alpha L_V(h_\alpha)$
4: Estimate generalization error of h_{α^*} over *test set*: $L_{\mathcal{D}}(h_{\alpha^*}) := L_T(h_{\alpha^*})$.
5: **return** $\alpha^*, h_{\alpha^*}, L_T(h_{\alpha^*})$
6: **end procedure**

The introduction of a third set of samples, the validation set, which is used to select the final model decouples the training- from the model selection- stages. This provides us with an unbiased estimator of the generalization error which we can also use to bound the generalization error.

Claim 6.2.1 Let $h_S := \mathcal{A}(S)$ be the hypothesis returned by a learner over the training sample S, and let V be a new ("fresh") *i.i.d* samples from $\mathcal{D}$. The empirical risk of h_S over V is an unbiased estimator of the generalization error of h_S.

Proof. Denote $m_V = |V|$. As the sample are identically distributed then:

$$\mathop{\mathbb{E}}_{V\sim\mathcal{D}^{m_V}}[L_V(h_S)] = \frac{1}{m_V}\sum_{i=1}^{m_V}\mathbb{E}_{(\mathbf{x}_i,y_i)\sim\mathcal{D}}[\ell(h_S,(\mathbf{x}_i,y_i))] = L_{\mathcal{D}}(h_S)$$

■

For the following, let us assume that the loss function is *bounded*. If the loss function is unbounded, the generalization error cannot be bounded. Suppose the loss function is bounded by 1.

Corollary 6.2.2 The generalization error of h_S is bound by:

$$\mathbb{P}\left[\left|L_V\left(h_S\right)-L_{\mathcal{D}}\left(h_S\right)\right| \leq \sqrt{\frac{\log\left(2/\delta\right)}{2m_V}}\right] \geq 1-\delta$$

Proof. Recall from Hoeffding's inequality that if $X_1,\ldots,X_m$ are a set of bounded iid random variables such that $0 \leq X_i \leq 1$ and $\overline{X} = \frac{1}{m}\sum X_i$, then: $\mathbb{P}\left[\left|\overline{X} - \mathbb{E}\left[\overline{X}\right]\right| \geq \varepsilon\right] \leq 2\exp\left(-2m\varepsilon^2\right)$. As V is a set of iid samples denote

$$Z_i \coloneqq (\mathbf{x}_i, y_i)$$

the random variable of the selected pair. Since the samples are iid then $Z_1,\ldots,Z_{m_V}$ are iid too. Next, denote

$$X_i \coloneqq \ell\left(h_S, Z_i\right)$$

the random variable of the loss over the Z_i sample. It holds that $X_1,\ldots,X_{m_V}$ are a set of iid random variables such that $0 \leq X_i \leq 1, \quad i = 1,\ldots,m_V$. Notice, that for $\overline{X} = \frac{1}{m_V}\sum X_i = L_V\left(h_S\right)$, as $L_V\left(h_S\right)$ is an unbiased estimator of the generalization error, we directly get that:

$$\mathbb{P}\left[\left|L_V\left(h_S\right)-L_{\mathcal{D}}\left(h_S\right)\right| \geq \varepsilon\right] \leq 2\exp\left(2m_V\varepsilon^2\right)$$

By setting $\varepsilon = \sqrt{\frac{1}{2m_V}\ln\frac{2}{\delta}}$ we conclude that:

$$\mathbb{P}\left[\left|L_V\left(h_S\right)-L_{\mathcal{D}}\left(h_S\right)\right| \leq \sqrt{\frac{\log\left(2/\delta\right)}{2m_V}}\right] \geq 1-\delta$$

■

6.2.2 Cross Validation

In reality, splitting our finite set of samples into three sets is problematic. In the great majority of cases we are unwilling to decrease the size of our training set. We have already seen that training over a smaller set yields inferior results. Therefore, designating an additional portion of our limited number of samples just for validation isn't feasible. Instead we would like to come up with some method to use S for training but still do proper model selection and -evaluation.

Potentially the simplest method to do so is cross-validation (CV) (Algorithm 10). Instead of thinking of S as a single set, let us think of it as a disjoint union of K equality sized sets, named folds. Then, for each $k = 1,\ldots,K$, we fit a model using all samples of S **except** samples belonging to the k'th fold. We then use the k'th fold to calculate the prediction error of the fitted model. Finally we report the estimated generalization error across all K folds. This method is called k-fold Cross Validation. The k'th fold, which is not used for training but only for evaluating the trained model functions as an unbiased estimator for the generalization error.

Algorithm 10 Cross-Validation

procedure k-FOLD-CROSS-VALIDATION(S, k $\mathcal{A}_\alpha$)
    Randomly partition S to k disjoint subsets $S = \biguplus_{i=1}^{k} S_i$.
    for $i = 1, \ldots, k$ **do**
        Train the model S except the i'th fold $S \setminus \{S_i\}$.
        Calculate the loss of the model on S_i functioning as a test set.
    end for
    return the estimated mean and standard deviation of the k losses obtained.
end procedure

Cross-Validation is the most popular method for choosing tuning parameters. For each candidate $\mathcal{A}_\alpha$, we train it k times according to the CV procedure, each time leaving out one of the k folds. Then we choose α (and as such the learner $\mathcal{A}_\alpha$) whose average error (over the k validation sets) was lowest. Once we finished the model selection phase using CV, and have a fully specified learner $\mathcal{A}$, we train $\mathcal{A}$ **again** on the entire training sample S. This way we are still able to train $\mathcal{A}$ on as many points as possible.

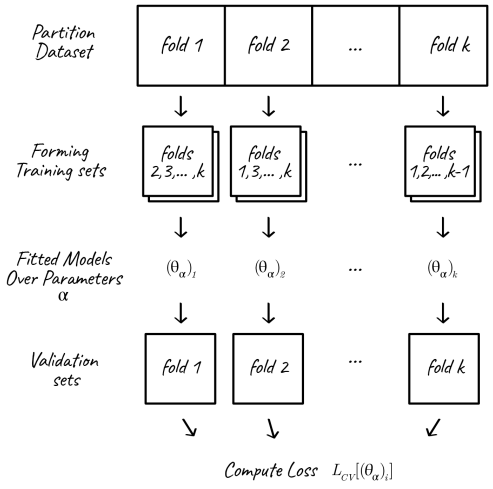


Figure 6.9: Schematics of Cross-Validation: *showing the partitioning of the dataset, training on all folds except the i'th fold to obtain estimators $(\theta_\alpha)_i$ and compute the cross-validation loss over the i'th fold left out.*

Cross validation is also used for model evaluation. Once we have a final, fully specified learner $\mathcal{A}$, we run k-fold CV and report the average error and standard deviation over the k folds. In this manner we supply both an estimation of the generalization error and a measure of accuracy for this estimate.

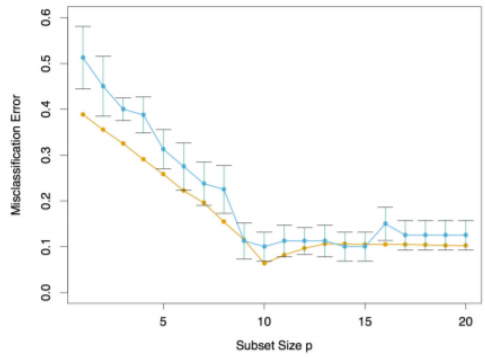


Figure 6.10: Generalization- and Cross-Validation errors as function of complexity: *Generalization error in orange and 10-fold cross-validation with errorbars in blue. Source: Elements of Statistical Learning*

Choosing k, the number of folds

By using CV we have now introduced another "hyper-hyper"-parameter. How should we choose the value for k? If $k = 1$ then there is no CV. If $k = 2$ we split the data into two equally sized folds where we train only one half of the data. Therefore, the reported CV error will be larger than the true generalization error (also known as out-of-sample error). If $k = m$, the sample size, it is called **leave-one-out** CV. Generally, if k is small we may be training on a dataset too small. As such the CV error may be biased upwards. On the other hand, if k is large, the training samples are very similar to each other. As such the trained models are highly correlated, which might introduce high variances.

R Another consideration when using CV is computational. Since we train each model k times, the larger k, the more computations we perform.

6.2.3 Bootstrap For Estimating Generalization Error

In subsection 5.3.1 we introduced the idea of Bootstrap, where by using re-sampling with replacement we can seemingly create new datasets. This method can also be used for estimating the generalization error using just the single training sample S. To do so let $\mathcal{A}_\alpha$ be some learner and $B \in \mathbb{N}$ the number of bootstrap samples to create. Very similar to the CV approach we could now:

- Draw a bootstrap sample $S^{(b)}$ by sampling m samples from S with replacement.
- By sampling $S^{(b)}$ we also obtain an independent test set $T^{(b)} := S \setminus S^{(b)}$ being the samples not chosen in the b-th bootstrap sample. These samples are also called the **out-of-bag** samples.
- Train $\mathcal{A}$ over $S^{(b)}$ to acquire an hypothesis h_S and test it over $T^{(b)}$.
- Finally, report the estimated mean and standard deviation of the generalization error, as measured over the B test sets.

6.2.4 Common Mistakes When Performing Model Selection

There are two very common mistakes when performing model evaluation using either of the methods seen before. The first causes over-estimation of the generalization error while the other causes under-estimation of the generalization error.

6.2.4.1 Over-estimating Generalization Error

Consider a family of learners $\{\mathcal{A}_\alpha\}$ over which cross-validation was used in order to determine α and obtain the fully specified learner $\mathcal{A}$. When each family member was trained it was done using a training set smaller than S. For k the number of folds used and m the number of samples in S, each model was trained using $m(k-1)/k$ samples.

Now, recall that successful learning depends on the number of training samples. When discussing PAC theory we defined the sample complexity as the minimal number of samples required to learn an hypothesis from a given hypothesis class, given some ε, δ. If m samples is a sufficient training size, but $m(k-1)k$ is not, we cannot guarantee the bounds over the generalization error. In such cases we will estimate the generalization error **too high**, namely over-estimate it.

To better determine the number of folds, we would like to have an hypothetical learning curve (Figure 6.11) for the model in question. Such a curve will capture the essence of the sample complexity function. Given a certain number of training samples the curve shows the expected success rate. Using think curve over a training set S with m samples, we can calculate the effective training set size when using k-fold cross-validation for a certain k. If the effective training size remains in the saturated area of the graph the estimated generalization error using cross-validation would not differ by much from the real generalization error. However, if the effective training size is where the slope of the graph is steep, we do not have a sufficient amount of training samples and will suffer from over-estimating the generalization error.

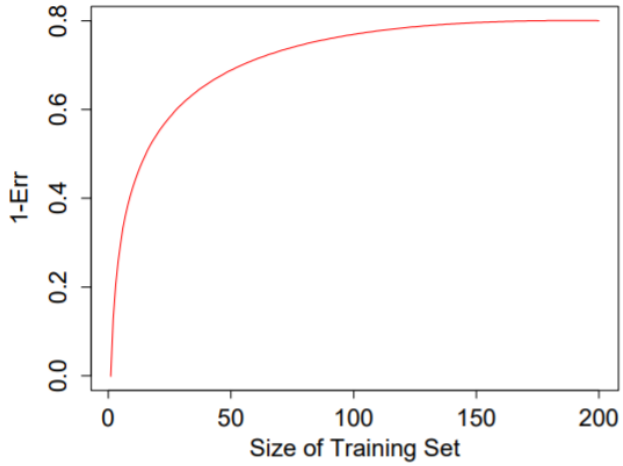


***Figure 6.11: Learning Curve of classifier**: showing the success $(1-Err)$ as a function of the training set size*

6.2.4.2 Under-estimating Generalization Error

A much more common mistake is being too optimistic in estimating the generalization error, causing under-estimation of it. Suppose we are given a learning problem and a dataset with m samples. We begin trying different types of models, each with its own tuning parameters. Perhaps we also alter the training sample S by removing- or adding features and addressing "problematic" samples. Eventually we have a "clean" training sample and a tuned model that works well. Now we perform model evaluation to estimate the generalization error of the chosen learner.

In such a scenario we will end up under-estimating the generalization error due to two mistakes:

- The first is known as **model snooping**. By trying out many learners, tuning the parameters of each and looking for one model that will perform very well, we slowly begin overfitting to the training sample. Each time we discarded some potential hypothesis class in favor of another that performed better we introduced more bias to the procedure. Even in the case where we use a validation set, if we evaluate the performance over it many times, we begin overfitting to it as well.
- The second is known as **data snooping**. Once we evaluate our selected model over a new test sample (or perhaps even when using it in production) this data did not under-go the same level of treatment. It might contain missing features or "problematic" samples that were dealt with in S. Therefore, the cross-validation procedure will under-estimate the generalization error over the new data.

To avoid this problem we should deal with these types of snooping. Limit model- and data-snooping to a small subset of S which will be "contaminated with optimism". Use it to get general understanding of the data and potential models. Only once the snooping stage has completed and a small set of candidate learners is chosen, use the entire training set for model selection- and evaluation. In addition, avoid manual data snooping. **Code the entire pre-processing stage**. At each iteration of Bootstap or cross-validation run the entire pre-processing step over the current sample, just like it would run when predicting over new data.

6.3 Summary and Exercises

1. Let $x_1,\ldots,x_m \in \mathbb{R}$ be a set of real values. Show that the minimizer μ of the sum of square distances $\sum (x_i-\mu)^2$ is given by the sample mean:

$$\frac{1}{m}\sum x_i = \underset{\mu\in\mathbb{R}}{\operatorname{argmin}} \sum (x_i-\mu)^2$$

2. Let $\mathbf{X} \in \mathbb{R}^{m\times d}$. Show that for all $\lambda > 0$ then $\mathbf{X}^\top\mathbf{X} + \lambda \cdot I_d$ is non-singular.
3. Let $\mathbf{X},\mathbf{y}$ be a regression problem with Gaussian errors $\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon$, $\varepsilon \sim \mathcal{N}\left(0,\sigma^2 I_m\right)$ and suppose $\mathbf{X}^\top\mathbf{X}$ invertible.
   - Show that $\widehat{\mathbf{w}}_\lambda^{ridge} = A_\lambda \hat{\mathbf{w}}$ where $A_\lambda = \left(\mathbf{X}^\top\mathbf{X} + \lambda I_d\right)^{-1}\left(\mathbf{X}^\top\mathbf{X}\right)$. Conclude that $\forall\lambda > 0$ then $\widehat{\mathbf{w}}_\lambda^{ridge}$ is a biased estimator for $\mathbf{w}$.
   - Recall that for any non random matrix B and a random vector $\mathbf{z}$ then $Var(B\mathbf{z}) = BVar(\mathbf{z})B^\top$. Show that the variance of the least squares estimator is given by $Var(\hat{\mathbf{w}}) = \sigma^2\left(\mathbf{X}^\top\mathbf{X}\right)^{-1}$ and the variance of the ridge solution is given by $Var(\widehat{\mathbf{w}}_\lambda^{ridge}) = \sigma^2 A_\lambda \left(\mathbf{X}^\top\mathbf{X}\right)^{-1} A_\lambda^\top$.
   - Show that the MSE of the ridge solution is lower than the MSE of the least squares solution.
4. Let $\mathbf{X},\mathbf{y}$ be a regression problem of orthogonal design and let ols be the least squares solution. Show that the the Ridge and Best-Subset estimator take the form of:

$$\begin{aligned}\widehat{\mathbf{w}}_\lambda^{subset} &:= \eta_{\sqrt{\lambda}}^{hard}\left(\widehat{\mathbf{w}}^{LS}\right)\\ \widehat{\mathbf{w}}_\lambda^{ridge} &:= \widehat{\mathbf{w}}^{LS}/\left(1+\lambda\right)\end{aligned}$$

7. Unsupervised Learning

Up to this point, our learning paradigm was of **supervised batch learning**. Given a sample space $\mathcal{X}$ and a label/response space $\mathcal{Y}$, we were interested in **prediction**: finding some way to predict $y \in \mathcal{Y}$ corresponding to a new unseen sample $x \in \mathcal{X}$, based on a training set of labeled samples $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$. However, there are many learning problems that do not fit into this framework of supervised batch learning. In one such set of problems we still consider a domain set $\mathcal{X}$ but there is no label/response space. Namely, the data is of the form $S = \{\mathbf{x}_i\}_{i=1}^m$ where $\mathbf{x}_i \in \mathcal{X}$. Such problems are called **unsupervised** learning problems, and the goal is to infer different properties of $\mathcal{X}$. A few examples are:

- **Uncovering low-dimensional structures:** In some cases we have reason to believe that some given data, though represented in high dimension, could actually be represented in a lower dimension. Consider for example the MNIST database of handwritten digits. This coprus of 28-by-28 pixels grey-scaled images shows scans of people handwriting of the digits $0, \ldots, 9$. Though each imange (sample) is represented in $\mathbb{R}^{28\times 28}$, there are very few variations on how people write digits. In Figure 7.1 we can see that in most images a big area in the surrounding of the image remains constant. This means that these pixels are not informative for predicting the digit. In addition, even though two images of the same digit (for example the digit 3 as seen below) show differences, they are still mostly the same. Therefore, it might be possible to **reduce the dimension** of the samples into a more compact, of lower dimension, representation.

- **Clustering:** Often, when we are given a dataset, we are interested in grouping (or segmenting) it into subsets such that those samples within each cluster are in some way more "closely related" to each other than to samples in other subsets. We refer to these subsets as *clusters*.

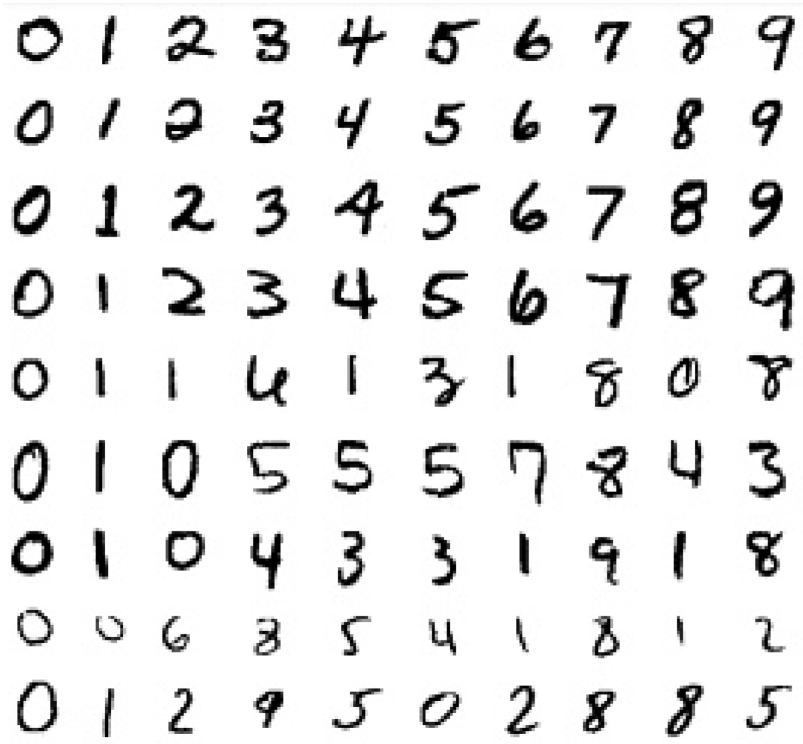


***Figure 7.1: Handwritten digits**: Liu et al., Handwritten digit recognition, Pattern Recognition 37(2) 2004*

Consider once more the MNIST digits dataset. In general, it seems logical that images of the same digit resemble each other more than images of different digits. If we would to try to cluster it (that is, segment its samples into different subsets) we would hope to get 10 separate subsets, each containing images of a specific digit.

- **Anomaly detection:** Suppose we are interesting in detecting when some given system is not behaving "as usual". Consider an air conditioning system or power plant. We place sensors in the system and would like to get a warning when something is behaving "strange". We do not know what exactly a "strange" behavior would look like, but it might be caused due to some mechanical malfunction, a software problem or even a cyber attack. We monitor the state of the system at all times. If we have installed d sensors, each time we take a reading of the system we get a sample $\mathbf{x}_i \in \mathbb{R}^d$. We train out learning system on the readings $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ where we believe these represent the normal state of the system. Now, given a new reading $\mathbf{x} \in \mathbb{R}^d$, is it "normal"? or is there something wrong? We are required to make a decision without have seeing the system in the "wrong"/"abnormal"/"strange" state before.

7.1 Dimensionality Reduction

Dimensionality reduction is the process of mapping some high dimensional space (the ambient space) to some new low dimensional space (the intrinsic space). Given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ we want to find some mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ for $k \ll d$, where $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)$ still resembles $\mathbf{x}_1, \ldots, \mathbf{x}_m$

in some sense. Different dimensionality reduction techniques are usually separated to linear- and non-linear techniques depending on the selected f.

There are different reasons to study and apply dimension reduction:

- **Learnability:** Some learning algorithms work better when the dimension d is small compared with the training sample size m, and might even fail if d is too large. Recall for example the case of linear regression. Though in both cases $m < d$ and $d < m$ we have a closed form solution, we have seen that if $d < m$ the results are numerically unstable.

- **Computation:** For many algorithm the time- and space complexity depends on the dimension d. By reducing the data dimensionality we can use fewer computational resources to perform the learning task we intended.

- **Visualization:** Visualization is an extremely useful tool both for explaining and exploring our data. If we can reduce dimension to $k \leq 4$ then we can plot the data (possibly on a 3 dimensional axis, with the forth dimension represented by color).

7.1.1 Principal Component Analysis (PCA)

Let $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ be some dataset and suppose that it approximately resides in some lower k-dimensional linear subspace of $\mathbb{R}^d$. We would like to find some linear transformation of the data to project it on that lower dimension subspace. To do so we need to solve two challenges:

1. There are infinitely many subspaces we could consider. Which subspace should we choose and how should we acquire it?

2. Even if we knew the subspace, projecting the data-points $\mathbf{x} \in \mathbb{R}^d$ onto the subspace does not yet reduce the dimension. We would want to find a way to represent each sample by the k coordinates of that sample *in* the subspace. This is also known as the *embedding* of the samples.

In the case of Principal Component Analysis (PCA), given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ we are searching for some linear transformation such that we minimize the squared error between the original samples and their transformation:

$$f^* := \underset{f}{argmin} \sum_{i=1}^{m} ||\mathbf{x}_i - f(\mathbf{x}_i)||^2 \tag{7.1}$$

This problem can be formulated in four different ways:

1. Finding the closest affine subspace to the points.
2. Finding the affine subspace that retains most of the variation seen in the data (sometimes referred to as *signal*).
3. Finding the affine subspace that minimizes the distortion of the pairwise distances between points in the ambient- compared to the intrinsic spaces.
4. Generalization of linear regression with Gaussian noise both in the explanatory variables directions and in the response direction. This interpretation is also known as probabilistic PCA.

Here, we will discuss the first two, most common, formulations.

7.1.1.1 Closest Affine Subspace

To simply the problem of finding the closest affine subspace, let us assume that the data is centered around the origin and as such instead of searching for some affine subspace, we are searching for an "ordinary" subspace. Therefore, we are searching for some linear map $W : \mathbb{R}^d \to \mathbb{R}^k$ and an "inverse" linear map $U : \mathbb{R}^k \to \mathbb{R}^d$

$$W^*, U^* = \underset{W \in \mathbb{R}^{k \times d}, U \in \mathbb{R}^{d \times k}}{\operatorname{argmin}} \sum_{i=1}^{m} ||\mathbf{x}_i - UW\mathbf{x}_i||^2 \tag{7.2}$$

Lemma 7.1.1 Let (U, W) be a solution to (7.2). Then U's columns are orthonormal and $W = U^\top$.

Proof. Let U, W be some matrices and consider the mapping $\mathbf{x} \mapsto (UW)\mathbf{x}$. The matrix (UW) is a d-by-d matrix of rank k, whos image is a k dimensional subspace of $\mathbb{R}^d$. Denote $S := Im(UW)$. As $\mathbf{x} \in \mathbb{R}^d$ and $(UW)\mathbf{x} \in S$, it holds that the projection that minimizes $||\mathbf{x} - UW\mathbf{x}||_2$ is the orthogonal projection onto the subspace. In addition, the point in S closest to $\mathbf{x}$, namely the orgothonal projection pf $\mathbf{x}$ onto S, is given by $VV^\top\mathbf{x}$ where the columns of V are an orthonormal basis of S:

$$\forall \mathbf{u} \in S \quad ||\mathbf{x} - \mathbf{u}||_2 \geq \left|\left|\mathbf{x} - VV^\top\mathbf{x}\right|\right|_2$$

Thus, the solution to 7.2 is U with orthonormal columns and $W := U^\top$. ■

Based on the above lemma, we can now write an equivalent problem to the PCA problem presented in 7.2. Observe that:

$$\begin{aligned}
\left|\left|\mathbf{x} - UU^\top\mathbf{x}\right|\right|^2 &= ||\mathbf{x}||^2 - 2\mathbf{x}^\top UU^\top\mathbf{x} + \mathbf{x}^\top UU^\top UU^\top\mathbf{x} \\
&= ||\mathbf{x}||^2 - \mathbf{x}^\top UU^\top\mathbf{x} \\
&\overset{(*)}{=} ||\mathbf{x}||^2 - \left(U^\top\mathbf{x}\right)^\top U^\top\mathbf{x} \\
&= ||\mathbf{x}||^2 - trace\left(U^\top\mathbf{x}\left(U^\top\mathbf{x}\right)^\top\right) \\
&= ||\mathbf{x}||^2 - trace\left(U^\top\mathbf{x}\mathbf{x}^\top U\right)
\end{aligned}$$

where $(*)$ is because $\forall \mathbf{v}, \mathbf{u} \in \mathbb{R}^k \; trace\left(\mathbf{u}\mathbf{v}^\top\right) = \mathbf{v}^\top\mathbf{u}$. As the trace is a linear operator we can re-write an equivalent problem:

$$U^* = \underset{U \in \mathbb{R}^{d \times k}, U^\top U = I}{argmax} \sum_{i=1}^{m} trace\left(U^\top\mathbf{x}_i\mathbf{x}_i^\top U\right) = \underset{U \in \mathbb{R}^{d \times k}, U^\top U = I}{argmax} \; trace\left(U^\top \sum_{i=1}^{m} \mathbf{x}_i\mathbf{x}_i^\top U\right) \tag{7.3}$$

Theorem 7.1.2 Let $A = \sum \mathbf{x}_i\mathbf{x}_i^\top$ and let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ be the k leading eigenvectors of A. Then, the solution to the PCA problem is given by $U \in \mathbb{R}^{d \times k}$ whose columns are $\mathbf{u}_1, \ldots, \mathbf{u}_k$.

Proof. Denote $A = \sum \mathbf{x}_i\mathbf{x}_i^\top$. Then, we need to solve

$$\underset{U \in \mathbb{R}^{d \times k}, U^\top U = I}{argmax} \; trace\left(U^\top AU\right)$$

Notice that A is a square symmetric matrix so let $A = VDV^\top$ be the EVD of A, where the diagonal of D are the eigenvalues of A in decreasing order $\lambda_1 \geq \ldots \geq \lambda_d$ and the columns of V are the corresponding eigenvectors. So:

$$\begin{array}{rcccl} trace\left(U^\top AU\right) & = & trace\left(U^\top VDV^\top U\right) & \overset{(*)}{=} & trace\left(B^\top DB\right) \\ & = & trace\left(DBB^\top\right) & = & \sum_{j=1}^{d}\left[DBB^\top\right]_{jj} \\ & = & \sum_{j=1}^{d}[D]_j\left[BB^\top\right]_{\cdot,j} & = & \sum_{j=1}^{d}\lambda_j\cdot\left[BB^\top\right]_{jj} \\ & = & \sum_{j=1}^{d}\lambda_j\cdot\sum_{i=1}^{k}B_{ji}^2 & & \end{array}$$

where $B = V^\top U \in \mathbb{R}^{d\times k}$. Notice that

$$B^\top B = U^\top VV^\top U = I_d$$
$$\Downarrow$$
$$\mathbb{1}[j\neq i] = \left\langle\left[B^\top\right]_j,[B]_{\cdot,i}\right\rangle = \left\langle[B]_{\cdot,j},[B]_{\cdot,i}\right\rangle$$

meaning that the columns of B are orthonormal, and therefore $\sum_{j=1}^{d}\sum_{i=1}^{k}B_{ji}^2 = k$. Based on B, define $\widetilde{B} = \left[\begin{array}{c|c} B & M \end{array}\right] \in \mathbb{R}^{d\times d}$ with $M \in \mathbb{R}^{d\times(d-k)}$ completing an orthonormal basis in R^d. If so, then $\forall j\ \sum_{i=1}^{d}\widetilde{B}_{ji}^2 = 1$, which implies that $\sum_{i=1}^{k}B_{ji}^2$. As such

$$trace\left(U^\top AU\right) = \sum_{j=1}^{d}\lambda_j\cdot\sum_{i=1}^{k}B_{ji}^2 \leq \max_{\beta\in[0,1]^d,||\beta||_1\leq k}\sum_{j=1}^{d}\lambda_j\beta_j = \sum_{j=1}^{k}\lambda_k$$

To conclude the proof, let U be the matrix whose columns are the k leading eigenvalues of A then

$$U^\top AU = \left[\begin{array}{c} u_1^\top \\ \vdots \\ u_k^\top \end{array}\right]A\left[\begin{array}{ccc} u_1 & \cdots & u_k \end{array}\right] = \left[\begin{array}{c} u_1^\top \\ \vdots \\ u_k^\top \end{array}\right]\left[\begin{array}{ccc} \lambda_1 u_1 & \cdots & \lambda_k u_k \end{array}\right] = diag\left(\lambda_1,\ldots,\lambda_k,\overbrace{0,\ldots,0}^{d-k}\right)$$

and $trace\left(U^\top AU\right) = trace\left(diag\left(\lambda_1,\ldots,\lambda_k,0,\ldots,0\right)\right) = \sum^k\lambda_i$, as requested. ■

Generalizing For Affine Subspaces

In the above theorem we in fact found the closest subspace- and not the closest affine subspace to the data. To find the closest affine subspace, we would like to allow the mapping W to be **affine**. To achieve this we generalize the above by considering $W : \mathbb{R}^d \to \mathbb{R}^k$ to be of the form

$$W(\mathbf{x}) \coloneqq \widetilde{W}(\mathbf{x}-\mu), \quad \mu\in\mathbb{R}^d, \widetilde{W}:\mathbb{R}^d\to\mathbb{R}^k$$

This allows us to "shift" the data before applying the linear map $\widetilde{W}$. When adding μ to the optimization problem above, we find that the minimizer μ is given by: $\mu = \frac{1}{m}\sum\mathbf{x}_i$. This is the empirical mean of the data, often denoted by $\overline{\mathbf{x}}$. So, in order to find the closest affine subspace to the data we center the matrix A:

$$A \coloneqq \sum_{i=1}^{m}(\mathbf{x}_i-\overline{\mathbf{x}})(\mathbf{x}_i-\overline{\mathbf{x}})^\top$$

yielding the following pseudo-code for the PCA algorithm:

The eigenvalues of A, $\lambda_1 \geq \ldots \geq \lambda_d$ are referred to as the **Principal Values** of $\mathbf{X}$. The eigenvectors of A, $\mathbf{u}_1,\ldots,\mathbf{u}_d$ are referred to as the **Principal Components**. Notice that the matrix A is a d-by-d positive semi-definite matrix. Therefore the PCA algorithm is in fact a case of a matrix diagonalization algorithm where we simply try to find the eigenvalues and eigenvectors of some target matrix.

Algorithm 11 PCA

procedure PCA($\mathbf{X}$, k) ▷ $\mathbf{X}$ The design matrix of m samples and d features
  Compute $A \leftarrow \sum_{i=1}^{m} (\mathbf{x}_i - \overline{\mathbf{x}}) (\mathbf{x}_i - \overline{\mathbf{x}})^\top$
  Let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ be the eigenvectors of A corresponding the largest eigenvalues.
  return $\mathbf{u}_1, \ldots, \mathbf{u}_k$
end procedure

7.1.1.2 Maximum Retained Variance

Another way to think about PCA is as a dimensionality reduction technique that maintains the maximum amount of variance of the data possible in a $k < d$ dimensional subspace. To prove so we will have to solve a constraint maximization problem using Lagrange Multipliers.

Theorem 7.1.3 Let $\mathbf{X}$ be an m-by-d design matrix. The projection of $\mathbf{X}$ onto a k dimensional linear subspace that retains maximum of the variance in $\mathbf{X}$ is given by the matrix $U \in \mathbb{R}^{d \times k}$ whose columns are the k eigenvectors with leading eigenvalues of the sample covariance matrix S.

Proof. We begin with considering the projection onto a one-dimensional subspace. Without loss of generality let $\mathbf{v} \in \mathbb{R}^d$ be a unit vector used to project the data onto. The expected value of the projected data is

$$\mathbb{E}_{\mathbf{x}} \left[\mathbf{v}^\top \mathbf{x} \right] = \frac{1}{m} \sum \mathbf{v}^\top \mathbf{x}_i = \mathbf{v}^\top \overline{\mathbf{x}}$$

Therefore, the variance of the projection is given by:

$$\begin{aligned} Var\left(\mathbf{v}^\top \mathbf{x}\right) &= \mathbb{E}_{\mathbf{x}} \left[\left(\mathbf{v}^\top \mathbf{x}_i - \mathbb{E}_{\mathbf{x}} \left[\mathbf{v}^\top \mathbf{x} \right] \right)^2 \right] &&= \frac{1}{m} \sum \left(\mathbf{v}^\top \mathbf{x}_i - \mathbf{v}^\top \overline{\mathbf{x}} \right)^2 \\ &= \frac{1}{m} \sum \left[\mathbf{v}^\top (\mathbf{x}_i - \overline{\mathbf{x}}) \right]^2 &&= \frac{1}{m} \sum \left[\mathbf{v}^\top (\mathbf{x}_i - \overline{\mathbf{x}}) \right] \left[\mathbf{v}^\top (\mathbf{x}_i - \overline{\mathbf{x}}) \right]^\top \\ &= \frac{1}{m} \sum \mathbf{v}^\top (\mathbf{x}_i - \overline{\mathbf{x}}) (\mathbf{x}_i - \overline{\mathbf{x}})^\top \mathbf{v} &&= \mathbf{v}^\top \left[\frac{1}{m} \sum (\mathbf{x}_i - \overline{\mathbf{x}}) (\mathbf{x}_i - \overline{\mathbf{x}})^\top \right] \mathbf{v} \\ &= \mathbf{v}^\top S \mathbf{v} \end{aligned}$$

So now let us maximize the projected variance with respect to $\mathbf{v}$:

$$\widehat{\mathbf{v}} = \underset{||v||=1}{argmax} \mathbf{v}^\top S \mathbf{v}$$

To solve this optimization problem we will use the Lagrange multipliers method with the constraint $g(\mathbf{v}) = 1 - \mathbf{v}^\top \mathbf{v}$. So the lagrangian is given by:

$$\begin{gathered} \mathcal{L} = \mathbf{v}^\top S \mathbf{v} + \lambda g(\mathbf{v}) \\ \Downarrow \\ \frac{\partial}{\partial \mathbf{v}} \mathcal{L} = 2S\mathbf{v} - 2\lambda \mathbf{v} = 0 \end{gathered}$$

Therefore the maximizer of $\mathbf{v}^\top \mathbf{v}$ must be an eigenvector of S. Notice that by left-multiplying the derivative by $\mathbf{v}^\top$ we find that the retained variance itself is given by:

$$\mathbf{v}^\top S \mathbf{v} = \lambda \mathbf{v}^\top \mathbf{v} \overset{||\mathbf{v}||=1}{=} \lambda$$

Thus, the maximal retained variance is the largest eigenvalue λ_1, achieved by $\mathbf{v} := \mathbf{u}_1$.

Next, let us find the direction that retains the second largest amount of variance. As we are looking for an orthogonal projection we add an additional constraint that this direction is orthogonal to $\mathbf{u}_1$:

$$\widehat{\mathbf{v}} = \underset{||\mathbf{v}||=1, \mathbf{v}^\top \mathbf{u}_1 = 0}{argmax} \mathbf{v}^\top S \mathbf{v}$$

As before, when solving the constraint optimization problem for $\mathbf{v}$ we find that the direction retaining the second largest amount of variance is $\mathbf{v} := \mathbf{u}_2$, with the amount of variance being λ_2. Proving by induction we get that $\forall k \leq d$, the k dimensional subspace that retains maximal variance of projecting $\mathbf{X}$ onto it, is given by the k leading eigenvectors of S. ■

7.1.1.3 Link Between Closest Subspace and Maximum Variance

In the above we have seen two interpretations of PCA: one as closest subspace and one as maximizing variance. To understand how the two are connected consider the dataset seen in Figure 7.2 and specifically the $\mathbf{x}_i$ data-point. Notice that by orthogonally projecting $\mathbf{x}_i$ onto $\mathbf{u}_1$ a right-angle triangle is formed where:

- The edge denoted by a is the size of the projection of $\mathbf{x}_i$ onto $\mathbf{u}_1$: $a := \left|\left|\mathbf{x}_i^\top \mathbf{u}_1\right|\right|$. This is the measure maximized in the maximum variance interpretation.
- The edge denoted by b is the distance between the original data-points $\mathbf{x}_i$ and its orthogonal projection onto $\mathbf{u}_1$: $b := ||\mathbf{x}_i - \mathbf{x}_i \mathbf{u}_1||$. This is the measure minimized in the closest subspace interpretation.
- The edge denoted by c is the size of $\mathbf{x}_i$: $c := ||\mathbf{x}_i||$.

As this is a right-angle triangle, we know from the Pythagorean theorem that $c^2 = a^2 + b^2$. Therefore, if we find a PCA solution that minimizes b (the closest subspace), we in fact find a solution that maximizes a. Similarly by finding a solution that maximizes a (maximal projected variance), we find a solution that minimizes b.

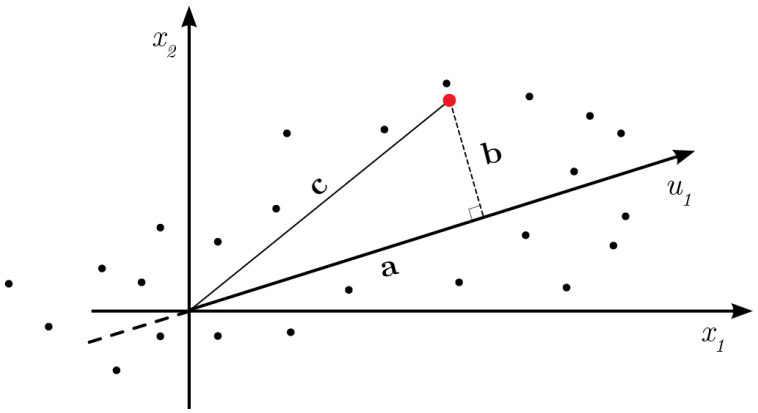


***Figure 7.2: Link Between PCA Interpretations**: Projection of $\mathbf{x}_i$ onto $\mathbf{u}_1$ forming a right-angle triangle.*

7.1.1.4 Projection- vs. Coordinates of Data-Points

When considering PCA (or many other dimensionality reduction algorithms), an important but often overlooked point is the difference between projection and embedding (i.e coordinates) of the data-points. Suppose $\mathbf{X} \in \mathbb{R}^{m \times d}$ and we run the PCA algorithm to find a lower k dimensional linear

subspace. The optimal PCA solution is the subspace that minimizes the sum of squared distances between each data-point $\mathbf{x}_i$ and its orthogonal projection onto the subspace. As we have seen above (7.1.2, 7.1.3), this subspace is spanned by the k leading eigenvectors of the $d \times d$ sample covariance matrix $S = \frac{1}{m}\sum (\mathbf{x}_i - \bar{x})(\mathbf{x}_i - \bar{x})^\top$.

This matrix $U \in \mathbb{R}^{d\times k}$ enables the **projection** of the points in $\mathbb{R}^d$ onto the k dimensional subspace, spanned by these leading eigenvectors. However, we would also like to actually reduce the dimension. Namely, find the map $W : \mathbb{R}^d \to \mathbb{R}^k$ and work with the dimension-reduced dataset $W(\mathbf{x}_1), \ldots, W(\mathbf{w}_m)$. As proven above, it is in fact $W = U^\top$ the map that provides the **embedding** of the data into the low dimension space. The vector $U^\top \mathbf{x}_i$ is a k dimensional vector, laying within the found k dimensional subspace. These are the **coordinates** of the original vector $\mathbf{x}_i$ according to the orthonormal set of k leading eigenvalues of S.

Let $\mathbf{u}_1, \ldots, \mathbf{u}_d$ be the d eigenvectors of S numbered in ascending order by the associated eigenvalues. As these vectors form a basis in $\mathbb{R}^d$, the vector $\mathbf{x}_i$ can be decomposed as $\mathbf{x}_i = \sum_{j=1}^{d} \langle \mathbf{x}_i, \mathbf{u}_j \rangle \mathbf{u}_j$.

- The **projecting** of $\mathbf{x}_i$ on the optimal k-dimensional subspace is given by $\mathbf{x}_i = \sum_{j=1}^{k} \langle \mathbf{x}_i, \mathbf{u}_j \rangle \mathbf{u}_j$.
- The **coordinates** (i.e. embedding) of $\mathbf{x}_i$ according to the k leading principal vectors are given by: $(\langle \mathbf{x}_i, \mathbf{u}_1 \rangle, \ldots, \langle \mathbf{x}_i, \mathbf{u}_k \rangle)^\top$.

In Figure 7.3 we illustrate the difference between the projection and the coordinates (embedding) of the data-points. This dataset was generated as follows: 1000 points were sampled from the ℓ_2 unit ball in $\mathbb{R}^2$. That is, $\{((x_i)_1, (x_i)_2)\}_{i=1}^{1000}$ where $(x_i)_1^2 + (x_i)_2^2 = 1$. Then Gaussian noise was added in a third axis:

$$\{((x_i)_1, (x_i)_2, \varepsilon_i)\}_{i=1}^{1000} \quad (x_i)_1^2 + (x_i)_2^2 = 1 \quad \varepsilon_1, \ldots, \varepsilon_{1000} \overset{i.i.d}{\sim} \mathcal{N}(0, 0.1)$$

Figure 7.3 (left) shows this dataset. Then, using PCA we first project the data onto the subspace defined by the 3 PCs (Figure 7.3, center). At this point each data-point is still represented in $\mathbb{R}^3$, but we can indeed see that it is mainly the first 2 axes that capture the signal in the data, with the third axis showing only minor variations (the added noise). Lastly, when embedding the data into a 2 dimensional subspace (Figure 7.3, right), we are left with the true signal of the data (i.e. samples drawn from a 2 dimensional ℓ_2 unit ball), and data-points are represented using only 2 coordinates.

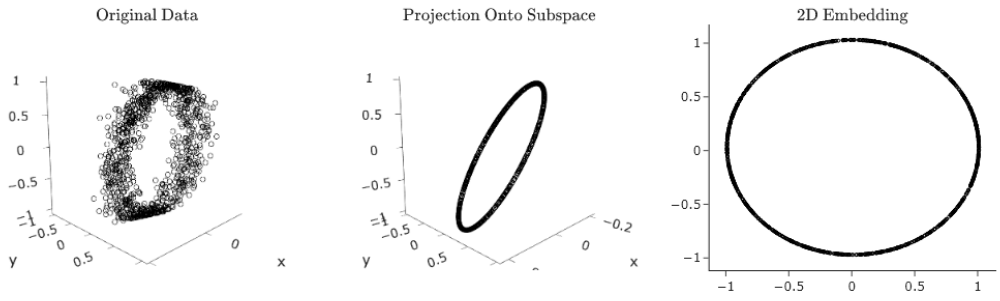


Figure 7.3: Projection vs. Embedding*: Dataset of sampels taken on the ℓ_2 unit ball with Gaussian noise in $\mathbb{R}^3$, and then rotated in a random direction. Chapter 7 Examples - PCA - Source Code*

7.1.1.5 Principal Components As "Typical Data-Points"

The principal components found by PCA are vectors in the ambient space $\mathbb{R}^d$. This means that they share the same dimension as the data-points $\mathbf{x}_1, \ldots, \mathbf{x}_m$. As they are orthonormal vectors chosen such that the first k provide the best linear approximation of dimension k to the dataset, we can look at them in an interesting way. They are, in this sense, the "typical" data-points, maximally different from each other (as they are an orthonormal set). Thus, it is often interesting to see what they represent as data-points: what part of the "signal" do they capture.

Returning to the MNIST Digits dataset of Figure 7.1 we can look at each principal component as a 28-by-28 image. Then, we can think of the representation of each image by the linear combination of these images:

$$\widehat{f}(\mathbf{x}) = \sum_{i=1}^{k} \alpha_i \mathbf{u}_i + \overline{\mathbf{x}}, \quad \alpha_1, \ldots, \alpha_k \in \mathbb{R}$$

where $\mathbf{u}_1, \ldots, \mathbf{u}_k$ are the leading k principal components and $\widehat{\mathbf{x}}$ is the centering vector of the data. Notice that as this is a linear combination we are reconstructing the sample $\mathbf{x}$ via adding and subtracting (weighted) principal components.

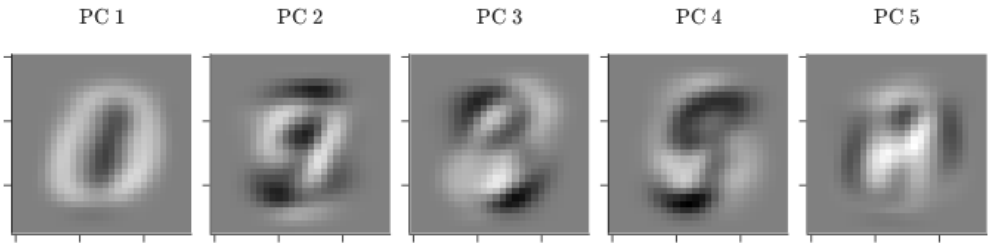


***Figure 7.4: Top Principal Components** of PCA fitted over the MNIST Digits dataset. Chapter 7 Examples - PCA - Source Code*

Figure 7.5 shows the reconstruction process of an image of digit 4 as the linear combination of the principal components. In each frame of the animation the restored image (center) is calculated as the restored image of the previous iteration plus $\alpha_i \mathbf{u}_i$ for $\mathbf{u}_i$ the principal component of the current iteration and α_i the loadings (coordinates) of the image for the i'th principal component.

7.2 Clustering

A very useful set of learning problems is of clustering. Often, either as part of data exploration or as part of the main analysis, we are interested in partitioning our data into **meaningful** groups. For example, given a corpus of images we might want to divide them into different groups such as nature/urban/people/etc. photographs; or, given the set of genes in the human genome, we might want to group together genes associated with specific diseases. In both these examples we are only given the samples (images or genes) but we do not have any **ground truth** (labels). We are not given the information of what is seen in each image, nor for each disease what genes are associated with it.

We would therefore want to define some measure of *similarity* between samples of a given domain.

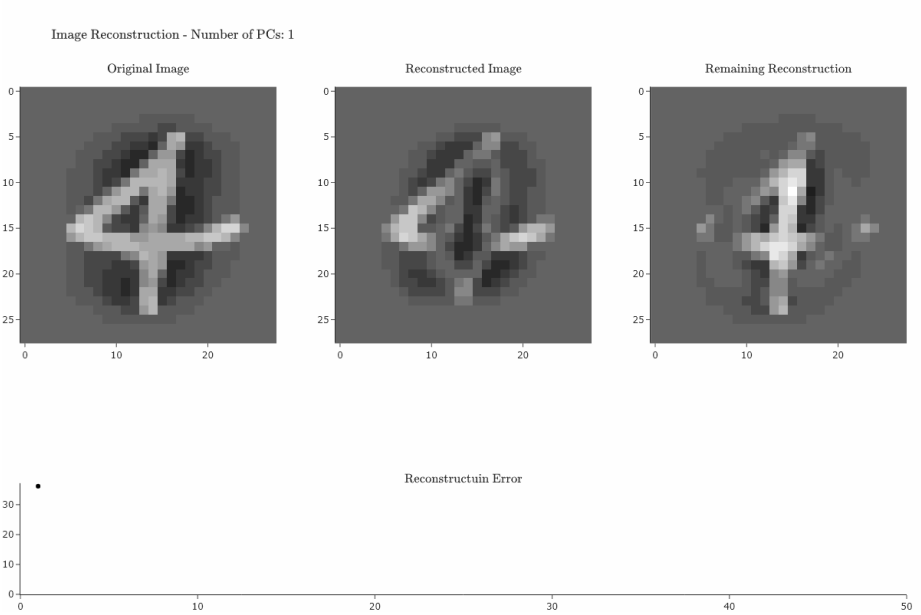


Figure 7.5: PCA Reconstruction: *Constructing image from principal components.* *Chapter 7 Examples - PCA - Source Code*

Using this similarity we could split the data into different subsets, where intuitively samples within a given set are *more similar* to one another compared to samples of different sets.

R This form of clustering, where we partition the data into distinct non-overlapping groups is also referred to as "hard assignment" as each sample is assigned to one specific subset. Sometimes, rather than assigning to some specific cluster we are interested in "partly assigning" to multiple clusters. This is referred to as "soft assignment/clustering".

7.2.1 K-Means

Though there are many different approaches to perform clustering, common to many is the notion of defining some representative data-point of the cluster. Then, clustering of data-points give the given sample is perform with respect to these cluster representatives.

Definition 7.2.1 A partition of a dataset $\{\mathbf{x}_i\}_{i=1}^m$ is a set $C_1,\dots,C_k$ such that $\{\mathbf{x}_i\}_{i=1}^m = \biguplus_{j=1}^k C_j$

Given some partition over the data and the representative data-points, we can define a **cost function** for the partition. For some metric (i.e. distance function) over the data $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ we define:

$$G_d(C_1,\dots,C_k,\mu_1,\dots,\mu_k) \coloneqq \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j} d(\mathbf{x},\mu_j) \tag{7.4}$$

where $\mu_1,\dots,\mu_k$ are the representatives of clusters $C_1,\dots,C_k$. Using this function, the goal is to find the partitioning that minimizes the following:

$$\{C_1,\dots,C_k\}^* = \underset{\{C_j,\mu_j\}_{j=1}^k}{argmin}\ G_d(C_1,\dots,C_k,\mu_1,\mu_k) = \underset{\{C_j,\mu_j\}_{j=1}^k}{argmin} \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j} d(\mathbf{x},\mu_j) \tag{7.5}$$

In the case of the k-means algorithm the cluster representatives are chosen to be:

$$\mu_j(C_j) \coloneqq \underset{\mu \in \mathcal{X}}{argmin} \sum_{\mathbf{x} \in C_j} d(\mathbf{x}, \mu)$$

In order to find the minimizers of 7.5 we would have to navigate the space of all possible partitions of m objects into k subsets. As there are an exponential amount of such subsets, minimizing the cost function G is NP-hard, and we must resort to **heuristics**. The most famous heuristic for minimizing G, when d is the Euclidean distance, is k-means.

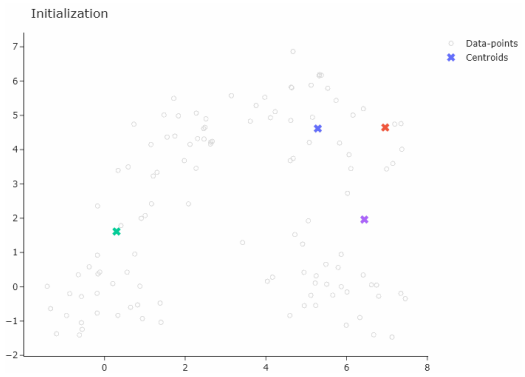
Algorithm 12 K-Means

procedure K-MEANS($\mathbf{X}$, k) ▷ $\mathbf{X}$ The design matrix of m samples and d features
  Choose initial centroids $\mu_1, \ldots, \mu_k$ randomly.
  while Not converged **do**
    Assignment: Assign each point $\mathbf{x}_i$ to the centroid closest to it:

$$C_j^{(t)} \coloneqq \left\{ \mathbf{x} | \mu_j = argmin_\mu \, d(\mathbf{x}, \mu), \mathbf{x} \in \mathbf{X} \right\}$$

    Update: Adjust cluster centroids by: $\forall j \in [k] \quad \mu_j^{(t+1)} \coloneqq \left| C_j^{(t)} \right|^{-1} \sum_{\mathbf{x} \in C_j^{(t)}} \mathbf{x}$
  end while
  return $C_1, \ldots, C_k$
end procedure

This clustering approach uses **Lloyd's Algorithm** to minimize G using an iterative strategy where each time we **alternate** between minimizing two terms. We first define a partition of the dataset, associating points to subsets by their nearest centroid. These subsets are called Voronoi cells. Then we re-calculate the cluster's centroid.



7.2.1.1 Convergence to Multiple- and Sub- Optimal Solutions

As clustering problems are NP-Hard, the K-Means algorithm algorithm uses the Lloyd's algorithm heuristic to find a good partition of the data. Being a heuristic, the optimality of the algorithm assignment to clusters isn't guaranteed. In fact, due to the nature of the random initialization of

centroids, the K-Means algorithm might converge into a sub-optimal solution or to there exists more than a single possible optimal solution.

Sub-optimality means that given a dataset $\{\mathbf{x}_i\}_{i=1}^m$ the K-Means algorithm returned some partitioning $C_1,\ldots,C_k$ such that there exists a different partitioning of the data $C'_1,\ldots,C'_k$ with a lower objective value: $G(C_1,\ldots,C_k) > G\left(C'_1,\ldots,C'_k\right)$. This is the product of the objective function not being convex, which means there could be several local minima, each achieving a different cost. In Figure 7.6 we see 4 groups of data-points and two initial centroids, positioned in such a way that forces the algorithm to converge into a sub-optimal solution. Optimal assignment is achieved for final centroids $\mu_i = (0,5)\,, \mu_j = (20,5)\,,\ i \neq j$.

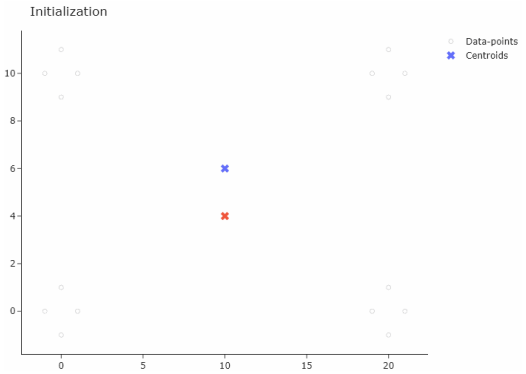


Figure 7.6: Suboptimal Solution: *Data-points and initial centroids leading to a suboptimal solution. Chapter 7 Examples - KMeans - Source Code*

Multiple optimal solution means that given a dataset $\{\mathbf{x}_i\}_{i=1}^m$ there are more than a single possible partitioning of the data that will yield the lowest objective value. It is important to note, that whenever discussing clustering assignments, we always look at the partitioning up to a permutation of the partition names. That is, suppose we are given the samples $1,2,3,4,5$ the partitioning of $C_1 = \{1,2,3\}\,, C_2 = \{4,5\}$ is identical to $C_1 = \{4,5\}\,, C_2 = \{1,2,3\}$, and clearly, both will achieve the same objective value. When referring to multiple optimal solution we mean:

$$\begin{array}{ll} \textit{Given } \{\mathbf{x}_i\}_{i=1}^m & \exists \{C_1\ldots,C_k\}\,, \left\{C'_1,\ldots,C'_k\right\} \\ \textit{For which} & G(C_1,\ldots,C_k) = G\left(C'_1,\ldots,C'_k\right) \\ \textit{It holds that} & \exists j \in [k]\ \ \forall l \in [k] \quad C_j \neq C'_l \end{array}$$

and that they achieve an objective value lower (or equal) to any other partitioning of the dataset. An example for such dataset and initialization of centroids is seen in Figure 7.6. In this case both centroids are initialized in the center of all data points.

7.2.1.2 Selection of k

As in different algorithms previously seen, in K-Means too we are required to provide a value for the hyper-parameter k. As we do not know how many clusters we really have in the dataset this is not a simple task. Notice that as long as we have more data-points than clusters, for any optimal solution with k clusters, we can achieve a solution with a lower objective for $k+1$. As such, simply running over different values of k and selecting the minimal value isn't a viable solution. Instead, we will apply a similar technique to the one used in PCA. We will plot the to objective achieved for

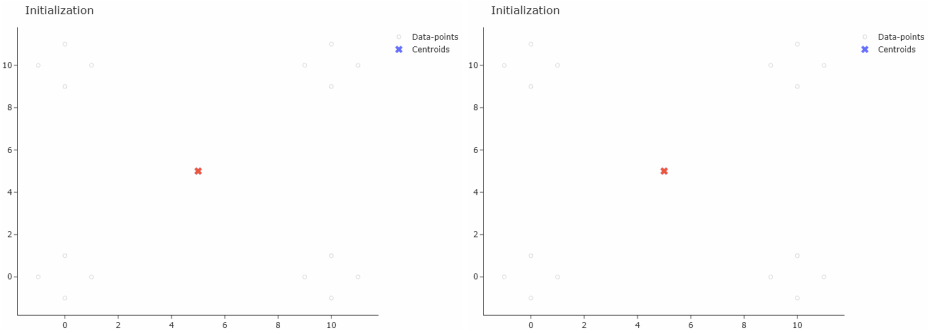


Figure 7.7: Multiple Optimal Solutions: *Data-points and initial centroids leading to two different optimal solutions. Chapter 7 Examples - KMeans - Source Code*

different values of k and select the value after which the improvement is less drastic. In Figure 7.8 we apply this strategy over the dataset seen in Figure **??**. As seen in the elbow-plot once we reach $k = 4$, the incremental improvement in score is much lower. Since the data was generated from 4 different Gaussians, this approach managed to find the correct value.

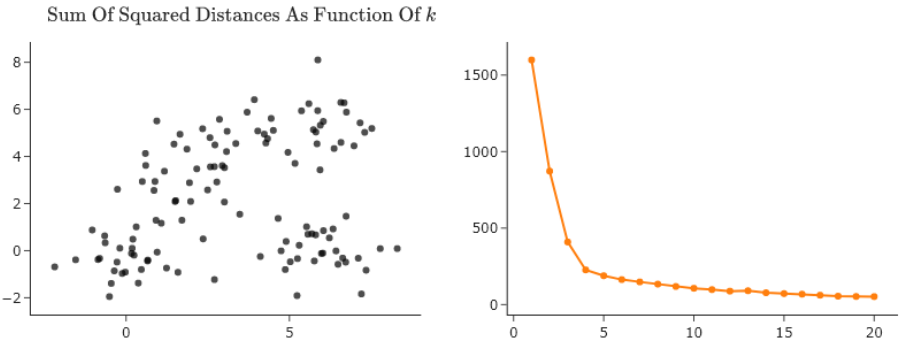


Figure 7.8: Selecting k hyper-parameter in K-Means *Chapter 7 Examples - KMeans - Source Code*

8. Kernel Methods

Recall the hypothesis classes of linear regression and classification:

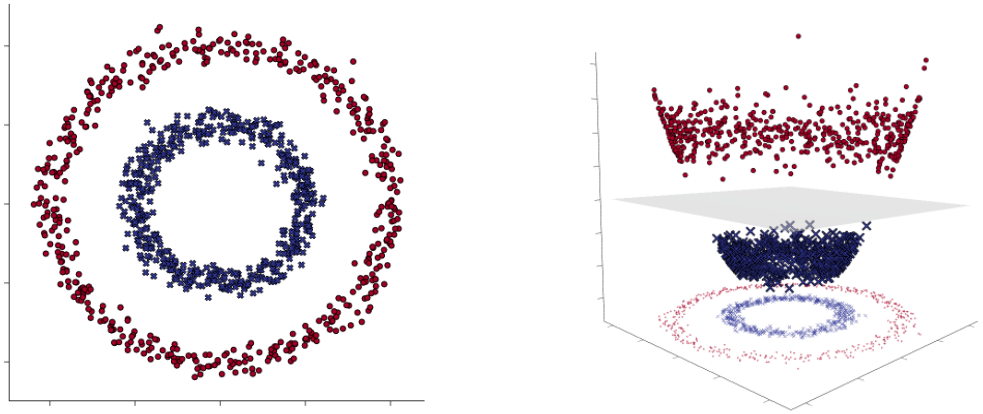
$$\begin{aligned} \mathcal{H}_{reg} &:= \left\{ \mathbf{x} \to w_0 + \mathbf{w}^\top \mathbf{x} \;\middle|\; w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d \right\} \\ \mathcal{H}_{reg} &:= \left\{ \mathbf{x} \to \text{sign}\left(w_0 + \mathbf{w}^\top \mathbf{x} \right) \;\middle|\; w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d \right\} \end{aligned} \tag{8.1}$$

These hypothesis classes are of limited power. Consider the case where $\mathcal{X} \subset \mathbb{R}^2$ with samples as seen in Figure 8.1a. In this case the dataset is not linearly separable and therefore algorithms matching the hypothesis class above will fail. Notice however, that if instead of using the original feature representation of the data, we embed the samples in a new feature space, such as mapping each sample $\mathbf{x} \to \left(x_1^2, x_2^2, x_1^2 + x_2^2\right)^\top$, we get a linearly separable sample in $\mathbb{R}^3$ (Figure 8.1b). Over this new representation we are able to separate the two labels using the SVM algorithm (found hyperplane represented in grey).

So, to enrich the expressiveness of an hypothesis class, we can consider embedding the data into another (potentially of higher dimension) feature space, over which we then learn some predictor. Schematically:

- Given some domain set $\mathcal{X} \subseteq \mathbb{R}^d$ and a learning algorithm $\mathcal{A}$, select an embedding $\psi : \mathcal{X} \to \mathcal{F}$ for some feature space $\mathcal{F}$. In many cases we will want $\mathcal{F} \subseteq \mathbb{R}^k$ for some $k \gg d$ (and possibly $k = \infty$).
- Train $\mathcal{A}$ over a training set $\{(\psi(\mathbf{x}_i), y_i)\}_{i=1}^m$ using the hypothesis classes:

$$\begin{aligned} \mathcal{H}_\psi &:= \left\{ \mathbf{x} \to w_0 + \mathbf{w}^\top \psi(\mathbf{x}) \;\middle|\; w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^k \right\} \\ \mathcal{H}_\psi &:= \left\{ \mathbf{x} \to \text{sign}\left(w_0 + \mathbf{w}^\top \psi(\mathbf{x}) \right) \;\middle|\; w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^k \right\} \end{aligned} \tag{8.2}$$



(a) *Two class dataset that is not linearly separable*

(b) *Dataset mapped to* $\mathbb{R}^3$ *using the mapping* $(x_1,x_2) \to \left(x_1^2, x_2^2, x_1^2 + x_2^2\right)^\top$

Figure 8.1: Illustration of mapping to Feature Space*: Originally linearly inseparable dataset is linearly separable in mapped feature space. Chapter 8 Examples - Source Code*

This approach seems very attractive as the assumption that our data can be described using some linear function (either for regression or classification) in *some* feature space does not look very limiting. The fact is that we have already encountered an example of such approach. In the case of polynomial fitting (section 2.2) given a sample in $x \in \mathbb{R}$ we first embedded it in some new feature space where $\psi(x)_i = x^i$ and then learn using the linear regression hypothesis class. Later in this chapter we will expand this mapping to use multivariate polynomials Lemma 8.2.2.

In general, in order to apply this strategy, we must address two challenges. The first is that we must find an appropriate mapping ψ under which it makes sense to learn using a linear predictor. The second is that we want our algorithm to be computationally efficient. When we map the samples to $\mathcal{F}$, as we are often interested in an embedding where $k \gg d$ and as ψ could be very complicated, computing $\psi(\mathbf{x})$ might be computationally expensive. We would like to find a way to select $h_S \in \mathcal{H}$ without explicitly evaluating the higher-dimensional features.

8.1 An Altered Learning Problem

We begin with noticing that many of the optimization problems presented in earlier chapters can be formulated as

$$\underset{\mathbf{w}\in\mathcal{F}}{\operatorname{argmin}} f\left(\langle \mathbf{w}, \psi(\mathbf{x}_1)\rangle, \ldots, \langle \mathbf{w}, \psi(\mathbf{x}_m)\rangle\right) + R(||\mathbf{w}||) \tag{8.3}$$

for f an arbitrary function and $R : \mathbb{R}_+ \to \mathbb{R}$ a monotonically non-decreasing function. For example, in the case of the Least Squares optimization problem f was the RSS function that given a prediction $\hat{y}_i$ returned $\sum(y_i - \hat{y}_i)^2$. In the case of the Soft-SVM optimization problem f calculated the mean hinge loss.

For optimization problems with the form as in (8.3) there exists an optimal solution that can be represented as a linear combination of the given samples $\psi(\mathbf{x}_i)$.

Theorem 8.1.1 — The Representer Theorem. Let $\mathcal{X}$ be a non-empty domain set and $\psi : \mathcal{X} \to \mathcal{F}$ be a mapping into some Hilbert space. Consider an optimization problem over $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$ of the form seen in (8.3) Then, there exists $\alpha \in \mathbb{R}^m$ such that $\mathbf{w}^* = \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i)$ is a minimizer of the optimization problem.

Proof. Let $\mathbf{w}^* \in \mathcal{F}$ be an optimal solution for the optimization problem. As $\mathcal{F}$ is a Hilbert space we can represent $\mathbf{w}^*$ as:

$$\mathbf{w}^* = \sum_{i=1}^m \alpha_j \psi(\mathbf{x}_i) + \mathbf{u}, \quad span(\psi(\mathbf{x}_1), \ldots, \psi(\mathbf{x}_m)) \otimes \mathbf{u} = \mathcal{F}$$

Denote $\mathbf{w} = \mathbf{w}^* - \mathbf{u}$ and notice that

$$\langle \mathbf{w}^*, \psi(\mathbf{x}_i) \rangle = \langle \mathbf{w}^* - \mathbf{u}, \psi(\mathbf{x}_i) \rangle = \langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle \quad i = 1, \ldots, m$$

Therefore both $\mathbf{w}^*$ and $\mathbf{w}$ obtain the same value for the fidelity term. Notice that since $||\mathbf{w}^*||^2 = ||\mathbf{w}||^2 + ||\mathbf{u}||^2 \geq ||\mathbf{w}||^2$ and R a monotonically non-decreasing function then $R(||\mathbf{w}||^*) \geq R(||\mathbf{w}||)$. Thus, put together, the objecting of $\mathbf{w}$ is bound from above by the objective of $\mathbf{w}^*$ and as such $\mathbf{w}$ is also an optimal solution. ■

Based on the representer theorem we can reformulate (8.3) with respect to α. By replacing $\mathbf{w}$ with the found representation that

$$\langle \mathbf{w}, \psi(\mathbf{x}_i) \rangle = \left\langle \sum_{j=1}^m \alpha_j \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \right\rangle = \sum_{j=1}^m \alpha_j \left\langle \psi(\mathbf{x}_j), \psi(\mathbf{x}_i) \right\rangle \quad i = 1, \ldots, m$$

and

$$||\mathbf{w}||^2 = \left\langle \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i), \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i) \right\rangle = \sum_{i,j=1}^m \alpha_i \left\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \right\rangle \alpha_j$$

By denoting $G \in \mathbb{R}^{m \times m}$ the Gram matrix whose entries are $G_{i,j} = \left\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \right\rangle$ we can rewrite (8.3) as

$$\underset{\alpha \in \mathbb{R}^m}{\operatorname{argmin}} \, f(G\alpha) + R\left(\alpha^\top G \alpha\right) \tag{8.4}$$

This dual representation of the former (primal) optimization problem searches for a solution $\alpha \in \mathbb{R}^m$ (i.e. the number of training samples) while the former optimization problem (8.3) searches for a solution $\mathbf{w} \in \mathcal{F}$ which can be arbitrary large (or as mentioned even of infinite dimension). In addition, notice that the dual problem is a quadratic problem in α and therefore, if G can be computed efficiently, then the entire problem can be solved efficiently. Once we have solved (8.4) for α we can derive the optimal solution for the primal optimization problem and predict the response of a new sample by:

$$\hat{y}(\mathbf{x}) = \langle \mathbf{w}^*, \psi(\mathbf{x}) \rangle = \left\langle \sum_{i=1}^m \alpha_i \psi(\mathbf{x}_i), \psi(\mathbf{x}) \right\rangle = \sum \alpha_i \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}) \rangle = \alpha^\top \mathbf{k} \tag{8.5}$$

where $k_i = \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}) \rangle$, $i = 1, \ldots, m$.

8.2 Characterizing Kernel Functions

In the section above we have shown that given some map $\psi : \mathcal{X} \to \mathcal{F}$ where $\mathcal{F}$ some Hilbert space we can solve the dual optimization problem with respect to $\alpha \in \mathbb{R}^m$ and then use α to predict the response of a given new sample.

However, as the mapped feature space might be of an arbitrary large dimension (or even of infinite dimension), computing the Gram matrix G or the predicted value of a new sample can be computationally expensive. To address this problem we will next apply the idea of *kernel substitution*. We will decide to use a (wide) specific family of functions, called *PSD kernel functions*, that even though might map to a very high dimension, can still be computed efficiently.

Definition 8.2.1 Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be some function. k is called a *kernel function* if k is symmetric, i.e. $\forall \mathbf{x}, \mathbf{x}' \in \mathcal{X}\ k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$.

Definition 8.2.2 Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel function. k is called a *Positive Semi-Definite kernel* if and only if for any $m \in \mathbb{N}$ and for any $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathcal{X}$ the matrix $K \in \mathbb{R}^{m \times m}$ whose entries are $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is a PSD matrix. K is referred to as the associated kernel matrix of k over $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

To check if a given function is a kernel function we could: (1) show the associated Gram matrix is PSD or (2) find the transformation ψ that satisfies that $\langle \psi(\mathbf{x}), (\mathbf{x}') \rangle$.

■ **Example 8.1** Consider the following function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for $\mathcal{X} = \mathbb{R}^d$ defined as $k(\mathbf{x}, \mathbf{x}') = 1,\ \mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Let us show that this is a PSD-kernel function as find a possible mapping function ψ such that $k(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), (\mathbf{x}') \rangle$.

Let $\mathbf{x}_1, \ldots \mathbf{x}_m \in \mathcal{X}$. We begin with showing that the associated Gram matrix of k over $\mathbf{x}_1, \ldots, \mathbf{x}_m$ is a PSD matrix. The Gram matrix of k over this set is: $G_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = 1$. Notice that as k is symmetric then G is a symmetric matrix. In addition, the eigenvalues of G are $m, 0 \geq 0$. Thus, $G \in PSD$ and k a valid PSD-kernel function.

As for finding a mapping function ψ, it must satisfy that $\langle \psi(\mathbf{x}), (\mathbf{x}') \rangle = 1$ for any $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$. So we could choose $\psi : \mathbb{R}^d \to \mathbb{R}^k$ that always returns some unit vector $\mathbf{v}$. Then $\langle \psi(\mathbf{x}), (\mathbf{x}') \rangle = \mathbf{v}^\top \mathbf{v} = ||\mathbf{v}|| = 1$. Notice that we have not specified the dimension of the feature space, which could be of any size we desire. ■

■ **Example 8.2** Consider the standard inner product in $\mathbb{R}^d$. Let us show that this is a PSD-kernel function. Let $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ and the matrix G whose entries are $G_{ij} = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$. Notice that as we can write G as $G = \mathbf{X}^\top \mathbf{X}$ where the i'th row of $\mathbf{X}$ is the i'th sample. Then G is a PSD matrix and thus k a PSD-kernel function. For this PSD-kernel function a possible mapping function can be the identity function $\psi(\mathbf{x}) \equiv \mathbf{x}$. ■

Interestingly, we are able to tie between the mapping functions ψ discussed in the previous section and PSD-kernel functions using the following (simplified) condition.

Theorem 8.2.1 — Mercer's Condition. Let $\psi : \mathcal{X} \to \mathcal{F}$ where $\mathcal{F}$ some Hilbert space. Then there exists a symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ implementing an inner product in $\mathcal{F}$ if and only if k is a PSD-kernel; namely, for all $\mathbf{x}_1, \ldots, \mathbf{x}_m$ $G_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$ and $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_i) \rangle$, the matrix G is PSD.

This condition tells us how we should choose ψ. We expressed the dual optimization problem (8.4) and the prediction (8.5) as *inner products* of the mappings of samples $\langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$. Therefore, if we choose ψ such that the Gram matrix $G_{ij} = \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j) \rangle$ is a PSD matrix then there exists a PSD-kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which yields the same outputs but is agnostic to the (high dimensional) feature space $\mathcal{F}$ and can be calculated efficiently.

8.2.1 The Polynomial- and Gaussian Kernel Functions

One commonly used kernel function is the polynomial kernel. It maps a given sample to the feature space whose coordinates correspond to all the monomials of degree at most k. This function extends the transformation seen in polynomial fitting (section 2.2).

▪ **Example 8.3** Before proving the general case let us consider the polynomial kernel in $\mathbb{R}^2$. Let $\mathbf{x}, \mathbf{x}' \in \mathcal{X} = \mathbb{R}^2$ with $x_0, x_0' = 1$ and $k = 2$ so:

$$\begin{aligned} (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^2 &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + 2x_1 x_1' + 2x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2' \end{aligned}$$

For ψ defined by $\psi(\mathbf{x}) = \left(1, 1 \cdot x_1, x_1 \cdot 1, 1 \cdot x_2, x_2 \cdot 1, x_1^2, x_2^2, x_1 \cdot x_2, x_2 \cdot x_1\right)^\top$ we achieve that $k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x})^\top \psi(\mathbf{x}')$. ▪

Claim 8.2.2 — Polynomial Kernels. Let $\mathcal{X} = \mathbb{R}^d$ and consider the function

$$k(\mathbf{x}, \mathbf{x}') := (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k$$

This is a valid PSD-kernel function corresponding the mapping:

$$\mathbf{x} \mapsto \left(1, \ldots, x_i, \ldots, \mathbf{x}_i \cdot \mathbf{x}_j, \ldots, \prod_{\substack{i \in J \\ J \subset [d], |J| = k}} x_i \right)$$

Proof. To show this is a valid kernel, we will find some mapping ψ such that $K(\mathbf{x}, \mathbf{x}')$ equals to $\psi(\mathbf{x})^\top \psi(\mathbf{x}')$. For simplicity let $x_0 = 1 = x_0'$. Then:

$$\begin{aligned} (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k &= \left(\textstyle\sum_{i=0}^d x_i x_i' \right)^k \\ &= \textstyle\sum_{J \in \{0, \ldots, d\}^k} \prod_{i=1}^k x_{J_i} x_{J_i}' \\ &= \textstyle\sum_{J \in \{0, \ldots, d\}^k} \prod_{i=1}^k x_{J_i} \prod_{i=1}^k x_{J_i}' \\ &= \textstyle\sum_{J \in \{0, \ldots, d\}^k} \psi(\mathbf{x})_J \, \psi(\mathbf{x}')_J \\ &= \psi(\mathbf{x})^\top \psi(\mathbf{x}') \end{aligned}$$

Where we define $\psi : \mathbb{R}^d \to \mathbb{R}^{(d+1)^k}$ such that each coordinate of $\psi(\mathbf{x})$ corresponds to some $J \in \{0, \ldots, d\}^k$ and is equal to $\prod_{i=1}^k x_{J_i}$. Therefore we obtained that $k(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ ■

Using the polynomial kernel and the associated mapping function ψ, let us return to the question of the cost of computing the optimization problem. If we were to solve the optimization problem as presented in (8.4) we would have to compute the Gram matrix G by calculating for every cell i,j the value $G_{ij} = \left\langle \psi(\mathbf{x}_i), \psi(\mathbf{X}_j) \right\rangle$. Even without considering computation times of the mappings themselves, the computation of the inner product in the mapped feature space would take $\mathcal{O}((d+1)^k)$ compared to $\mathcal{O}(d)$ in the optimization problem prior to the mapping. However, if we indeed do apply the *kernel substitution* idea and rather than explicitly compute the mapping (and thus the inner product in the feature space) only compute the kernel function then computing $(1+\langle \mathbf{x}, \mathbf{x}' \rangle)^k$ for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ is an $\mathcal{O}(d)$ operation. This improvement in computation time holds also when performing predictions.

R The polynomial kernel function presented above maps to all monomials of degree *at most* k. It can be shown that the function $k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle^k$ is also a valid PSD-kernel and maps to the space of all monomials of degree *exactly* k.

Another powerful and commonly used kernel function is the Gaussian kernel.

Claim 8.2.3 — Gaussian Kernels. The following is a valid kernel function:

$$k(\mathbf{x}, \mathbf{x}') := \exp\left(-\frac{1}{2\sigma^2} \left|\left| \mathbf{x} - \mathbf{x}' \right|\right|^2 \right), \quad \sigma^2 \in \mathbb{R}_+$$

with the mapping function ψ defined such that:

$$\forall n \in \mathbb{N} \quad \psi(\mathbf{x})_n := \frac{1}{\sqrt{n!}} \exp\left(-x^2/2\sigma\right) x^n$$

Proof. Let us begin with showing that $k(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x})^\top \psi(\mathbf{x}')$:

$$\begin{aligned} \psi(\mathbf{x})^\top \psi(\mathbf{x}') &= \sum_{n=0}^{\infty} \left(\frac{1}{\sqrt{n!}} \exp\left(-x^2/2\sigma\right) x^n \right) \left(\frac{1}{\sqrt{n!}} \exp\left(-(\mathbf{x}')^2/2\sigma \right) (\mathbf{x}')^n \right) \\ &= \exp\left(-\frac{\mathbf{x}^2 + (\mathbf{x}')^2}{2\sigma} \right) \sum_{n=0}^{\infty} \frac{(\mathbf{x}\mathbf{x}')^n}{n!} \\ &= \exp\left(-\frac{1}{2\sigma^2} \left|\left| \mathbf{x} - \mathbf{x}' \right|\right|^2 \right) \end{aligned}$$

■

Consider the feature space of the primal optimization problem for which we use the Gaussian kernel. Notice that as ψ maps to a space of infinite dimension, the primal optimization problem is one over infinitely many parameters. Using the dual representation however, we are able to find an equivalent optimal solution over a space with m parameters. In terms of computing the elements seen in the dual representation, by kernel substitution we understand that we do not have to explicitly compute $\psi(\mathbf{x}), \psi(\mathbf{x}')$ and there inner product in the feature space but only calculate the kernel function - an operation performed in $\mathcal{O}(d)$.

R The Gaussian kernel function is a specific case of the wider Radial Basis Function (RBF) kernel $k(\mathbf{x}, \mathbf{x}') := \exp\left(-\beta \left|\left| \mathbf{x} - \mathbf{x}' \right|\right|^2 \right)$.

8.2.2 Closure Properties For PSD-Kernels

Since a function is a valid PSD-kernel function if the associated Gram matrix is PSD we can derive several closure properties for PSD-kernel functions. Here are some of these properties.

Claim 8.2.4 Let k be some valid kernel function then the following are valid kernel functions:

1. $k(x,y)x^\top Ay$, where $A \in PSD$.
2. $c \cdot k_1(x,y)$ where $c > 0$.
3. $exp(k_1(x,y))$
4. $f(x)k_1 f(y)$

Using these closure properties we are able to construct new PSD-kernel functions and derive a very rich family of functions.

■ **Example 8.4** Using the closure properties above, let us show that the Gaussian kernel function is indeed a valid PSD-kernel.

$$\begin{aligned} k(\mathbf{x},\mathbf{x}') &= \exp\left(-||\mathbf{x}-\mathbf{x}'||^2/2\sigma^2\right) \\ &= \exp\left(-||\mathbf{x}||^2/2\sigma^2\right)\cdot\exp\left(-\mathbf{x}^\top\mathbf{x}'/\sigma^2\right)\cdot\exp\left(-||\mathbf{x}'||^2/2\sigma^2\right) \\ &\overset{(*)}{=} f(\mathbf{x})\exp\left(-\mathbf{x}^\top\mathbf{x}'/\sigma^2\right)f(\mathbf{x}') \end{aligned}$$

where we define $f(\mathbf{x}) = \exp\left(-||\mathbf{x}||^2/2\sigma^2\right)$. Now notice that this is a valid kernel as: (1) $\mathbf{x}^\top\mathbf{x}'$ is a valid kernel. (2) scaling by $\frac{1}{\sigma^2}$ is a valid kernel. (3) exponent of a valid kernel is a valid kernel. Finally, (4) multiplying by $f(\mathbf{x}), f(\mathbf{x}')$ from left and right is a valid kernel. ■

8.3 Kernelized Algorithms

Based on the Representer Theorem (8.1.1), if an optimization problem is in the specified form, we can derive a kernelized version of the algorithm by replacing $\mathbf{x}$ with $\psi(\mathbf{x})$, $\mathbf{w}$ with α and solve the optimization problem with respect to α. Below are a few examples of algorithms covered in previous chapters but adapted to a kernelized version.

8.3.1 Kernel Ridge Regression

Recall the Ridge Regression optimization problem (6.1.2.2) which jointly minimizes the RSS and the ℓ_2 norm of the coefficients vector $\mathbf{w}$:

$$\underset{\mathbf{w}\in\mathbb{R}^d}{\operatorname{argmin}} ||\mathbf{y}-\mathbf{X}\mathbf{w}||^2 + \lambda||\mathbf{w}||^2$$

Let us derive the dual representation of the optimization problem by replacing $\mathbf{x}$ with $\psi(\mathbf{x})$, $\mathbf{w}$ with α and solving for α. For the fidelity term then:

$$\begin{aligned} \sum_{i=1}^m (y_i - \langle\psi(\mathbf{x}_i),\mathbf{w}\rangle) &= \sum_{i=1}^m \left(y_i - \left\langle\psi(\mathbf{x}_i),\sum_{j=1}^m \alpha_j\psi(\mathbf{x}_j)\right\rangle\right) \\ &= \sum_{i=1}^m \left(y_i - \sum_{j=1}^m \alpha_j\left\langle\psi(\mathbf{x}_i),\psi(\mathbf{x}_j)\right\rangle\right)^2 \\ &= \sum_{i=1}^m \left(y_i^2 + 2y_i\sum_{j=1}^m \alpha_j\left\langle\psi(\mathbf{x}_i),\psi(\mathbf{x}_j)\right\rangle + \left(\sum_{j=1}^m \alpha_j\left\langle\psi(\mathbf{x}_i),\psi(\mathbf{x}_j)\right\rangle\right)^2\right) \end{aligned}$$

and for the regularization term then:

$$\begin{aligned} \lambda||\mathbf{w}||^2 &= \lambda\left(\sum_{i=1}^m \alpha_j\psi(\mathbf{x}_i)\right)^2 \\ &= \lambda\sum_{i,j=1}^m \alpha_i\left\langle\psi(\mathbf{x}_i),\psi(\mathbf{x}_j)\right\rangle\alpha_j \end{aligned}$$

which in matrix notation yields the following optimization problem:

$$\underset{\alpha\in\mathbb{R}^m}{\operatorname{argmin}} ||\mathbf{y}||^2 + 2y^\top G\alpha + \alpha^\top G^2\alpha + \lambda\,\alpha^\top G\alpha$$

Denote the function by f. We equate the gradient of f with respect to α to zero:

$$\begin{gathered}\nabla_\alpha f(\alpha) = 2G^2\alpha - 2G\mathbf{y} + 2\lambda G\alpha = 0 \\ \Downarrow \\ G(G+\lambda I_m)\alpha = G\mathbf{y}\end{gathered}$$

Since G is a PSD matrix then $G+\lambda I_m$ is a PD matrix and is invertible. Thus, the minimizer of the dual problem is given by

$$\hat{\alpha} = (G+\lambda I_m)^{-1}\,\mathbf{y}$$

Using this minimizer we can now express the prediction over a new sample $\mathbf{x}$ by:

$$\begin{aligned}\hat{y}(\mathbf{x}) &= \langle \mathbf{w}, \psi(\mathbf{x})\rangle \\ &= \textstyle\sum_{i=1}^m \alpha_i \langle \psi(\mathbf{x}_i), \psi(\mathbf{x})\rangle \\ &\overset{(*)}{=} \mathbf{k}^\top\alpha \\ &= \mathbf{k}^\top (G+\lambda I_m)^{-1}\,\mathbf{y}\end{aligned}$$

where $k_i = \langle \psi(\mathbf{x}_i), \psi(\mathbf{x})\rangle$. Notice that both G and the expression derived in the prediction only access the samples though inner-products of their mappings. Thus, we can simply replace the inner-products with the easy-to-compute PSD kernel function and therefore solve this optimization problem efficiently.

8.3.2 Kernel Regularized Logistic Regression

Similar to the Kernel Ridge Regression algorithm, we can derive a kernelized version for the regularized Logistic Regression classifier. For simplicity let the regularization function be the ℓ_2 norm of $\mathbf{w}$. Then, the objective of the ℓ_2-regularized Logistic Regression is

$$f(\mathbf{w}) = \sum_{i=1}^m \left[\log\left(1+e^{\langle \mathbf{x}_i,\mathbf{w}\rangle}\right) - y_i\langle \mathbf{x}_i,\mathbf{w}\rangle\right] + \lambda\,||\mathbf{w}||^2$$

By replacing $\mathbf{x}$ with $\psi(\mathbf{x})$ and $\mathbf{w}$ with α we derive the following objective function:

$$\begin{aligned}f(\alpha) &= \textstyle\sum_{i=1}^m \left[\log\left(1+e^{\langle \psi(\mathbf{x}_i), \sum_j \alpha_j\psi(\mathbf{x}_j)\rangle}\right) - y_i\langle \psi(\mathbf{x}_i), \sum_j \alpha_j\psi(\mathbf{x}_j)\rangle\right] + \lambda\,\alpha^\top G\alpha \\ &= \textstyle\sum_{i=1}^m \left[\log\left(1+e^{\sum_j \alpha_j\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j)\rangle}\right) - y_i\sum_j \alpha_j\langle \psi(\mathbf{x}_i), \psi(\mathbf{x}_j)\rangle\right] + \lambda\,\alpha^\top G\alpha \\ &= \textstyle\sum_{i=1}^m \log\left(1+e^{[G\alpha]_i}\right) - \sum_{i=1}^m y_i\,[G\alpha]_i + \lambda\,\alpha^\top G\alpha \\ &= \textstyle\sum_{i=1}^m \log\left(1+e^{[G\alpha]_i}\right) - \mathbf{y}^\top G\alpha + \lambda\,\alpha^\top G\alpha\end{aligned}$$

To find the minimizer we equate the derivative with respect to each coordinate of α to zero:

$$\begin{aligned}\frac{\partial f(\alpha)}{\partial \alpha_j} &= \textstyle\sum_{i=1}^m \frac{\partial \log\left(1+e^{[G\alpha]_i}\right)}{\partial \alpha_j} - \frac{\partial \mathbf{y}^\top G\alpha}{\partial \alpha_j} + \lambda\,\frac{\partial \alpha^\top G\alpha}{\partial \alpha_j} \\ &= \textstyle\sum_{i=1}^m \frac{1}{1+\exp(-[G\alpha]_i)}\cdot\frac{\partial e^{[G\alpha]_i}}{\partial \alpha_j} - [G\mathbf{y}]_j + 2\lambda\,[G\alpha]_j \\ &= \textstyle\sum_{i=1}^m \frac{\exp([G\alpha]_i)}{1+\exp(-[G\alpha]_i)}\cdot\frac{\partial [G\alpha]_i}{\partial \alpha_j} - [G\mathbf{y}]_j + 2\lambda\,[G\alpha]_j \\ &= \textstyle\sum_{i=1}^m G_{ij}\cdot\frac{\exp([G\alpha]_i)}{1+\exp(-[G\alpha]_i)} - [G\mathbf{y}]_j + 2\lambda\,[G\alpha]_j \quad = \quad 0\end{aligned}$$

Which can be written as follows:

$$\sum_{i=1}^{m} G_{ij}\left(\frac{1}{1+\exp\left(-\left[G\alpha\right]_i\right)}+2\lambda\alpha_i\right)=\sum_{i=1}^{m} G_{ij}y_i$$

Therefore the minimizers $\alpha_1,\ldots,\alpha_m$ are the solutions to the following transcendental equations

$$\frac{1}{1+\exp\left(-\left[G\alpha\right]_i\right)}+2\lambda\alpha_i=y_i$$

8.3.3 Kernel PCA

Another algorithm that can be kernalized is the PCA algorithm and is known as the Kernel PCA (Schölkopf *et al.,* 1998). In this algorithm we solve the PCA problem in some feature space $\mathcal{F}$, rather than using the original coordinates. To show we can apply the Kernel Trick to the PCA algorithm we must first show that we can re-write PCA such that accessing the data is done only through inner products of the samples. In addition we would need to show that projecting samples onto the found subspace can be done only through inner products. Then, if we are able to do so, we can replace $\mathbf{x}$ with $\psi(\mathbf{x})$ substitute the inner product with the associated PSD kernel function.

Let $\mathbf{X}\in\mathbb{R}^{m\times d}$ be a *centered* design matrix. We saw that when we solve the PCA optimization problem we are in fact solving an eigenvalues problem for the sample covariance matrix $C=\mathbf{X}^\top\mathbf{X}=\sum_i \mathbf{x}_i\mathbf{x}_i^\top$ (Lemma 7.1.2). We would like to represent this sum of outer-products via inner-products.

Let $\mathbf{v}$ be an eigenvector of C corresponding to eigenvalue $\lambda\neq 0$ so $\lambda\mathbf{v}=C\mathbf{v}$. This means that $\mathbf{v}\in Im(C)$ and therefore is in the span of $\mathbf{x}_1,\ldots,\mathbf{x}_m$. As such, solving an eigenvalue problem for C is equivalent to finding a vector $\mathbf{v}$ for which :

$$\lambda\left\langle\mathbf{x}_i,\mathbf{v}\right\rangle=\left\langle\mathbf{x}_i,C\mathbf{v}\right\rangle\quad i=1,\ldots,m \tag{8.6}$$

In addition, by the definition of C notice that $\lambda\mathbf{v}=C\mathbf{v}=\sum_i\mathbf{x}_i\mathbf{x}_i^\top\mathbf{v}=\sum_i\left\langle\mathbf{x}_i,\mathbf{v}\right\rangle\mathbf{x}_i$ and therefore $\mathbf{v}=\sum_i\frac{1}{\lambda}\left\langle\mathbf{x}_i,\mathbf{v}\right\rangle\mathbf{x}_i$. By denoting $\alpha_i=\frac{1}{\lambda}\left\langle\mathbf{x}_i,\mathbf{v}\right\rangle$ we get that $\mathbf{v}=\sum_{i=1}^m\alpha_i\mathbf{x}_i=\mathbf{X}^\top\alpha$. Now, rather than solving the equations in (8.6) notice the following. Fix some $i\in[m]$ and then:

$$\begin{aligned}
\lambda\left\langle\mathbf{x}_i,\textstyle\sum_j\alpha_j\mathbf{x}_j\right\rangle &= \left\langle\mathbf{x}_i,C\left(\textstyle\sum_j\alpha_j\mathbf{x}_j\right)\right\rangle\\
&= \textstyle\sum_j\alpha_j\left\langle\mathbf{x}_i,C\mathbf{x}_j\right\rangle\\
&= \textstyle\sum_j\alpha_j\left\langle\mathbf{x}_i,\sum_l\mathbf{x}_l\mathbf{x}_l^\top\mathbf{x}_j\right\rangle\\
&= \textstyle\sum_j\alpha_j\sum_l\left\langle\mathbf{x}_l,\mathbf{x}_j\right\rangle\left\langle\mathbf{x}_i,\mathbf{x}_l\right\rangle\\
&\Downarrow\\
\lambda\textstyle\sum_j\alpha_j\left\langle\mathbf{x}_i,\mathbf{x}_j\right\rangle &= \textstyle\sum_j\alpha_j\sum_l\left\langle\mathbf{x}_l,\mathbf{x}_j\right\rangle\left\langle\mathbf{x}_i,\mathbf{x}_l\right\rangle
\end{aligned}$$

Which in matrix notation is $\lambda G\alpha=G^2\alpha$ where we define G to be the Gram matrix over $\mathbf{x}_1,\ldots,\mathbf{x}_m$: $G_{ij}=\mathbf{x}_i^\top\mathbf{x}_j$, or equivalently $G=\mathbf{X}\mathbf{X}^\top$. For eigenvectors of G, not corresponding to zero eigenvalues the solution of $\lambda G\alpha=G^2\alpha$ is equivalent to that of $\lambda\alpha=G\alpha$.

Therefore, once we find $\alpha^{(1)},\ldots,\alpha^{(m)}$ the eigenvectors of G we can then:

- Obtain the eigenvectors of C by $\mathbf{v}^{(l)}=\sum_{i=1}^m\alpha_i^{(l)}\mathbf{x}_i$.
- Project the data-points on the low dimension subspace by:

$$\widetilde{\mathbf{x}}_l:=\left\langle\mathbf{v}^{(l)},\mathbf{x}\right\rangle=\sum_i\alpha_i^{(l)}\left\langle\mathbf{x}_i,\mathbf{x}\right\rangle$$

As we were able to represent the original PCA problem and the projection operating using inner-products of samples we can now apply the Kernel Trick to the PCA and substitute the inner-products with some PSD kernel function. Thus, for a feature map ψ where $C = \sum_i \psi(\mathbf{x}_i)\psi(\mathbf{x}_i)^\top$:

- Solve the eigenvalue problem $\lambda\alpha = G\alpha$, for $G_{ij} = \langle \psi(x_i), \psi(\mathbf{x}_j) \rangle$.
- Project the data-points by:

$$\widetilde{\mathbf{x}}_l := \left\langle \mathbf{v}^{(l)}, \psi(\mathbf{x}) \right\rangle = \sum_i \alpha_i^{(l)} \langle \psi(\mathbf{x}_i), \psi(\mathbf{x}) \rangle = \sum_i \alpha_i^{(l)} k(\mathbf{x}_i, \mathbf{x})$$

Centering Matrix In Feature Space

The above derivation assumes the given design matrix $\mathbf{X}$ is centered. Unlike in the case of PCA, in the Kernel PCA we cannot compute the mean and reduce it as it would require to compute $\forall i\ \phi(\mathbf{x}_i)$. Therefore, we would need to use a different approach:

Lemma 8.3.1 Let G be the Gram matrix of ψ over the dataset $\mathbf{X}$. The centered Gram matrix to be used in the Kernel PCA algorithm is given by:

$$\widetilde{G} = G - \mathbf{1}_m G - G\mathbf{1}_m + \mathbf{1}_m G \mathbf{1}_m$$

Proof. Denote $\widetilde{\psi}(\mathbf{x})$ the projected data-point after centering:

$$\widetilde{\psi}(\mathbf{x}) = \psi(\mathbf{x}) - \frac{1}{m}\sum_l \psi(\mathbf{x}_l)$$

where $\mathbf{1}_m \in \mathbb{R}^{m\times m}$, $[\mathbf{1}_m]_{ij} = 1/m$. So the elements of the centered Gram matrix are given by:

$$\begin{aligned}
\widetilde{G}_{ij} &= \langle \widetilde{\psi}(\mathbf{x}_i), \widetilde{\psi}(\mathbf{x} - j) \rangle \\
&= \left\langle \psi(\mathbf{x}_i) - \tfrac{1}{m}\textstyle\sum_l \psi(\mathbf{x}_l), \psi(\mathbf{x}_j) - \tfrac{1}{m}\sum_k \psi(\mathbf{x}_k) \right\rangle \\
&= \psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_j) - \tfrac{1}{m}\textstyle\sum_k \psi(\mathbf{x}_i)^\top \psi(\mathbf{x}_k) - \tfrac{1}{m}\sum_l \psi(\mathbf{x}_l)^\top \psi(\mathbf{x}_j) + \tfrac{1}{m^2}\sum_{l,k} \psi(\mathbf{x}_l)^\top \psi(\mathbf{x}_k) \\
&= G_{ij} - \tfrac{1}{m}\textstyle\sum_l G_{il} - \tfrac{1}{m^2}\sum_k G_{jk} + \tfrac{1}{m}\sum_{l,k} G_{lk}
\end{aligned}$$

which in matrix notation is $\widetilde{\psi}(\mathbf{x}) = \psi(\mathbf{x}) - \frac{1}{m}\sum_l \psi(\mathbf{x}_l)$. ■

Put all together, the pseudo-code for the Kernel PCA algorithm is:

Algorithm 13 Kernel-PCA

procedure KERNEL-PCA($\mathbf{X}$, k, l) ▷ k the kernel computing ϕ and l the dimension to reduce to
Compute the centered Gram matrix (8.3.1):

$$\widetilde{K} = K - \mathbf{1}_m K - K\mathbf{1}_m + \mathbf{1}_m K \mathbf{1}_m, \quad K_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$$

Let $\alpha^{(1)}, \ldots, \alpha^{(l)}$ be the eigenvectors of $\widetilde{K}$ corresponding the largest eigenvalues.
Compute the corresponding eigenvectors of A by: $\mathbf{v}^{(j)} := \sum_{i=1}^m \alpha_i^{(j)} \mathbf{x}_i$
return $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(l)}$
end procedure

R The PCA algorithm seen in Algorithm 11 diagonalizes the sample covariance matrix, which is a d-by-d matrix and has a time complexity of $\mathcal{O}\left(d^3\right)$. If $d \gg m$ this can be computationally expensive. Though the proof of the Kernel PCA algorithm we have actually shown that we can solve the "normal" PCA by computing $\mathbf{X}\mathbf{X}^\top$ instead of using $\mathbf{X}^\top\mathbf{X}$. This means that time complexity is reduced to $\mathcal{O}\left(m^3\right)$ for computing $\mathbf{X}\mathbf{X}^\top$ and then $\mathcal{O}\left(m^2 \cdot d\right)$ for adjusting the eigenvalues of $\mathbf{X}\mathbf{X}^\top$ to those of $\mathbf{X}^\top\mathbf{X}$.

8.4 Summary and Exercises

Kernelization is a powerful learning concept through which rather than defining more expressive hypothesis classes we find a new embedding of the data. Through the use of the Representer Theorem (8.1.1) we have seen that given an algorithm formulated as (8.3)

$$\underset{\mathbf{w}\in\mathcal{F}}{\operatorname{argmin}}\, f\left(\left\langle \mathbf{w}, \psi(\mathbf{x}_1)\right\rangle, \ldots, \left\langle \mathbf{w}, \psi(\mathbf{x}_m)\right\rangle\right) + R(||\mathbf{w}||)$$

there exists a coefficients vector $\alpha \in \mathbb{R}^m$ such that $\mathbf{w} = \sum \alpha_i \psi(\mathbf{x}_i)$ for which α is the minimizer of the dual optimization problem (8.4)

$$\underset{\alpha\in\mathbb{R}^m}{\operatorname{argmin}}\, f\left(G\alpha\right) + R\left(\alpha^\top G\alpha\right)$$

Then, over the dual representation, we learned from Mercer's Condition (8.2.1) that we do not have to specify ψ explicitly. Instead we can specify a PSD kernel function k for which there exists a mapping ψ to some Hilbert space. We know that such kernel satisfies that $k(\mathbf{x},\mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}')\rangle \;\forall \mathbf{x},\mathbf{x}' \in \mathcal{X}$. By doing so, and substituting the inner-products of the mappings with k we are able to compute the kernel's output without explicitly evaluating the function in the high dimensional feature space. This means that we are able to efficiently solve an optimization problem for which the dimension of the primal representation is of arbitrary size.

Exercises

1. Prove Mercer's condition for $\mathcal{X} := \mathbb{R}^d$, $\mathcal{F} := \mathbb{R}^k$, $d,k \in \mathbb{N}$. That is, let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and let $\psi : \mathbb{R}^d \to \mathbb{R}^k$. k is a PSD-kernel function if and only if $k(\mathbf{x},\mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}')\rangle$.
2. Let $k(\mathbf{x},\mathbf{x}') = \langle \mathbf{x},\mathbf{x}'\rangle^k$. Show that k is a valid PSD-kernel function mapping to the feature space of all monomials of degree at most k. In addition what is the dimension of such feature space?