

Introduction to Machine Learning

Course Book

Chapters 1-3

INTRODUCTION TO MACHINE LEARNING

Matan Gavish and Gilad Green



Contents

0.1	Preface	7
0.1.1	Notation	7
1	Mathematical Basis	9
1.1	Linear Algebra	9
1.1.1	Linear Transformations	9
1.1.2	Norms, Inner Products and Projections	11
1.1.3	Matrix Decompositions	14
1.2	Multivariate Calculus	17
1.2.1	Derivatives, Gradients and Jacobians	17
1.2.1.1	Derivatives	17
1.2.1.2	Gradients	18
1.2.1.3	Jacobians	19
1.2.1.4	Chain Rules	20
1.2.2	Function Approximations	22
1.2.2.1	First Order Approximation	22
1.2.2.2	Second Order Approximation	23
1.2.3	Convexity	24
1.2.3.1	Convex Sets and Functions	24
1.2.3.2	High Order Conditions For Convexity	26
1.3	Probability and Statistics	27
1.3.1	Fundamental Definitions	28
1.3.1.1	Probability Space	28
1.3.1.2	Random Variables	29
1.3.1.3	Mean and Variance	30

1.3.2	A Little Statistics: Mean and Variance Estimation	32
1.3.3	Multivariate Probabilities	33
1.3.3.1	Normal Distribution	34
1.3.3.2	Covariance Matrix	36
1.3.3.3	Linear Transformations of the Data Set	37
1.3.4	Probability Inequalities	39
1.3.4.1	Markov's and Chebyshev's inequalities	40
1.3.4.2	Coin Prediction Example	42
2	Linear Regression	47
2.1	Regression Models	47
2.1.1	Linear Regression	48
2.1.2	Designing A Learning Algorithm	49
2.1.2.1	Realizability	49
2.1.2.2	Loss Function	50
2.1.2.3	Empirical Risk Minimization	50
2.1.2.4	Least Squares	50
2.1.2.5	The Normal Equations	51
2.1.3	Numerical Considerations When Implementing	55
2.1.4	A Statistical Model - Adding Noise	57
2.1.4.1	The Maximum Likelihood principle	58
2.2	Polynomial fitting	59
2.2.1	Bias and Variance of Estimators	61
2.3	Summary and Exercises	63
3	Classification	65
3.1	Classification Overview	65
3.1.1	Loss Functions	67
3.1.2	Type-I and Type-II Errors	67
3.1.3	Measurements of performance	68
3.1.4	Decision Boundaries	69
3.1.5	Studying A New Classifier	69
3.2	Half-Space Classifier	70
3.2.1	Learning Linearly Separable Data Via ERM	72
3.2.2	Computational Implementation	72
3.2.3	Convex Optimization	72
3.2.4	Solving ERM for Half-Spaces	73
3.2.4.1	The Perceptron Algorithm	74
3.2.5	Learner ID Card	74
3.3	Support Vector Machines (SVM)	75
3.3.1	Maximum Margin Learning Principle	75
3.3.2	Hard-SVM	76
3.3.2.1	Solving Hard-SVM	77

3.3.3	Soft-SVM	79
3.3.4	Learner ID Card	81
3.4	Logistic Regression	81
3.4.1	A Probabilistic Model For Noisy Labels	81
3.4.1.1	The Hypothesis Class	83
3.4.1.2	Learning Via Maximum Likelihood	83
3.4.2	Computational Implementation	84
3.4.3	Interpretability	84
3.4.4	Predictions Over New Samples & The ROC Curve	84
3.4.5	Learner ID Card	86
3.5	Nearest Neighbors	87
3.5.1	Prediction Using k-NN	87
3.5.2	Selecting Value of k Hyper-Parameter	88
3.5.3	Computational Implementation	89
3.5.4	Learner ID Card	89
3.6	Decision Trees	89
3.6.1	Axis-Parallel Partitioning of \mathbb{R}^d	90
3.6.2	Classification & Regression Trees	91
3.6.3	Growing a Classification Tree	92
3.6.4	CART Heuristic For Growing Trees	93
3.6.5	Pruning a Decision Tree	95
3.6.6	Interpretability	95
3.6.7	Learner ID Card	95

0.1 Preface

0.1.1 Notation

Field/Context	Symbol	Meaning
General	\mathbb{R}	The set of real numbers
Mathematics	\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
	$\{0, \dots, m\}$	The set of natural numbers between 0 and m
	$[m]$	The set of natural numbers between 1 and m
	$ S $	The number of elements in the set S
	$a := b$	Defining a as the expression 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}
	$\ \cdot\ _1$	$= \sum_{i=1}^d x_i $ (the ℓ_1 norm of \mathbf{x})
	$\ \cdot\ _2$ or $\ \cdot\ $	$= \sqrt{\sum_{i=1}^d x_i^2}$ (the ℓ_2 norm of \mathbf{x})
	$\ \cdot\ _\infty$	$= \max_i x_i $ (the ℓ_∞ norm of \mathbf{x})
	$A_{i,j}$	The j th element of the i th row of A
	A^\top	The transpose of A : $A_{i,j}^\top = A_{j,i}$
	$\det(A)$ or $ A $	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 i th 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}
	$\stackrel{!}{=}$	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 probability 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 Learning	m	Number of samples
	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
	$S = x_1, \dots, x_n$	A sequence of samples from domain set
	$S = z_1, \dots, z_n$	A sequence of samples and responses from product space \mathcal{Z}
	$x_1, \dots, x_m \stackrel{iid}{\sim} \mathcal{D}$	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
	$\hat{\theta}$	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 $T : V \rightarrow W$ 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 $T : V \rightarrow W$. We define the *four fundamental subspaces*:

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

- Image- (or column/range-) space of A as $Im(A) := \{w \in W | w = Ax, x \in V\}$. Also denotes as $Col(A)$ or $\mathcal{R}(A)$.
- Row space of A as $Im(A^\top) := \{x \in V | x = A^\top w, w \in W\}$. Equivalently it can be defined as the column space of A^\top and therefore denoted as $Col(A^\top)$.
- Null space of A^\top as $Ker(A^\top) := \{x \in W | A^\top x = 0\}$. 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:

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

Let us rearrange the equations in matrix form. Given a linear equation we will denote all its 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:

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

■

(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 x s and the y s, 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 $X \subseteq \mathbb{F}^k$ $d : X \times X \rightarrow \mathbb{R}_+$ 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 $\|\cdot\| : \mathbb{R}^d \rightarrow \mathbb{R}_+$ 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: $\|av\| = |a| \cdot \|v\|$.

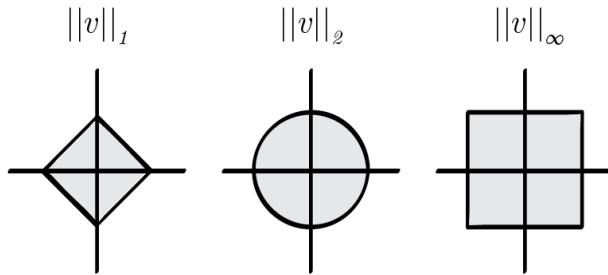
- Triangle inequality: $\|v + u\| \leq \|v\| + \|u\|$.

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

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

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 $\langle \cdot, \cdot \rangle : V \times V \rightarrow \mathbb{R}$ 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 $\|\cdot\| : H \rightarrow \mathbb{R}_+$ 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:

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

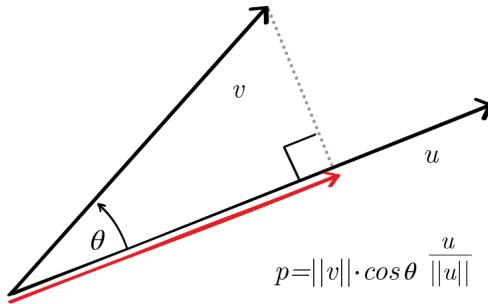
Hence, we conclude that:

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

■

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, \dots, 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, \dots, 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, \dots, v_k are the eigenvectors of P which correspond to the eigenvalue 1.
- $(I - P)P = 0$
- $\forall x \in \mathbb{R}^d$ and $\forall u \in V$, $\|x - u\| \geq \|x - Px\|$
- $x \in V \Rightarrow Px = x$



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 $u_1, \dots, u_n \in \mathbb{R}^d$ 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 Value Decomposition (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 values 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 $\sigma_1 \geq \dots \geq \sigma_d \geq 0$. The diagonal elements σ_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 values 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 $\text{Ker}(A)$.
- 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, if all singular values are strictly larger than zero ($\sigma_d > 0$) then $\dim(\text{Ker}(A)) = 0$ 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 Σ^\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 $\text{rank}(A) = r$. 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):

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

Since $\sigma_{r+1}, \dots, \sigma_m$ 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:

$$A = \tilde{U}\tilde{\Sigma}\tilde{V}^\top = \overbrace{\left[\begin{array}{c|c} | & | \\ u_1 & \cdots & u_r \\ | & & | \end{array} \right]}^{m \times r} \overbrace{\left[\begin{array}{ccc} \sigma_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sigma_r \end{array} \right]}^{r \times r} \overbrace{\left[\begin{array}{c|c} - & v_1^\top \\ \vdots & \vdots \\ - & v_r^\top \end{array} \right]}^{r \times d} \quad (1.1)$$

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 are connected to one another. The following lemma relates 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 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 $\sigma_1^2, \dots, \sigma_d^2$ 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$. 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:

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

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 $f : \mathbb{R} \rightarrow \mathbb{R}$. The derivative of f at point $x \in \mathbb{R}$ is defined as

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

■ Example 1.2 — ReLU. Consider the Rectified Linear Unit 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 of 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$. The partial derivative of f at point $\mathbf{x} \in \mathbb{R}^d$ with respect to x_i is defined as

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

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 $f(\mathbf{x}) = \max_i(x_1, \dots, x_d)$ the partial derivatives of f at x_i are:

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

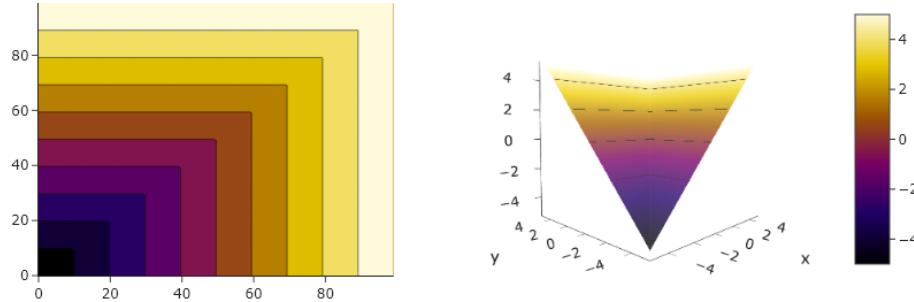


Figure 1.1: Visualization of bi-variate max function in 3D and 2D. [Chapter 1 Examples - Source Code](#)

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

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

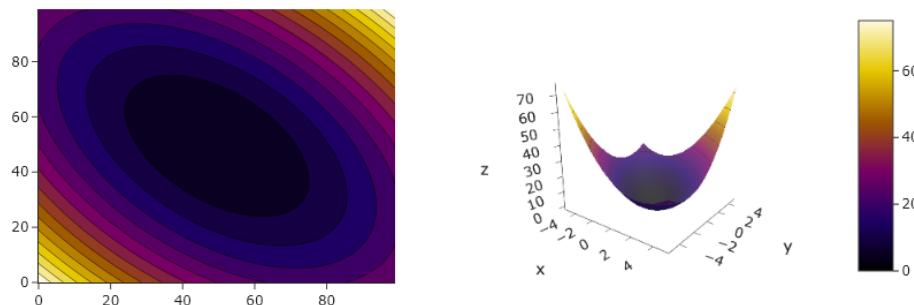


Figure 1.2: Visualization of bi-variate polynomial of degree 2 in 3D and 2D. [Chapter 1 Examples - Source Code](#)

1.2.1.2 Gradients

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

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

■ **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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ 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:

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

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

■

Exercise 1.4 — Norm. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $f : \mathbb{R}^m \rightarrow \mathbb{R}^d$.

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 extrema points of this function. In this scenario, the described function gets four parameters and returns two outputs $f : \mathbb{R}^4 \rightarrow \mathbb{R}^2$. In order to find the extrema points we need to generalize the gradient definition to vector-valued functions.

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

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

■ **Example 1.6** Let us revisit 1.5 where $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ 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, \dots, 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 $f : \mathbb{R} \rightarrow \mathbb{R}$ and $g : \mathbb{R} \rightarrow \mathbb{R}$ be two differential functions, then the derivative of the composite $f \circ g$ is:

$$(f \circ g)' := (f' \circ g) \cdot g'$$

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

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

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

In element-wise notation:

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

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 seen in 1.5 that $J_{\mathbf{x}}(g) = A$. Notice, that as $\text{Im}(f) \subseteq \mathbb{R}$, 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:

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

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 $S : \mathbb{R}^d \rightarrow [0, 1]^d$ 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 beginning 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 $f : \mathbb{R} \rightarrow \mathbb{R}$ 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 $f : \mathbb{R} \rightarrow \mathbb{R}$, 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $f : \mathbb{R}^2 \rightarrow \mathbb{R}$, 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ 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 $f(x_1, x_2) = x_1^2 + x_1 x_2 + x_2^2$. 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:

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

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:

$$f(\mathbf{p} + \mathbf{p}_0) \approx f(\mathbf{p}_0) + \nabla f(\mathbf{p}_0)^\top \mathbf{p} + \frac{1}{2} \mathbf{p}^\top H_f(\mathbf{p}_0) \mathbf{p} \quad (1.3)$$

■ **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:

$$\begin{aligned} f(\mathbf{p}_0 + \mathbf{p}) &\approx f(\mathbf{p}_0) + \nabla f(\mathbf{p}_0)^\top \mathbf{p} \\ &= x_0^2 + x_0 y_0 + y_0^2 + \begin{bmatrix} 2x_0 + y_0 \\ 2y_0 + x_0 \end{bmatrix}^\top \begin{bmatrix} x \\ y \end{bmatrix} \\ &= x_0^2 + x_0 y_0 + y_0^2 + 2x_0 x + y_0 x + 2y_0 y + x_0 y \end{aligned}$$

Now, let us add the second order approximation:

$$\begin{aligned} f(\mathbf{p}_0 + \mathbf{p}) &\approx x_0^2 + x_0 y_0 + y_0^2 + 2x_0 x + y_0 x + 2y_0 y + x_0 y + \frac{1}{2} \begin{bmatrix} x \\ y \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \\ &= x_0^2 + x_0 y_0 + y_0^2 + 2x_0 x + y_0 x + 2y_0 y + x_0 y + x^2 + yx + y^2 \\ &= (x_0 + x)^2 + (x_0 + x)(y_0 + y) + (y_0 + y)^2 \end{aligned}$$

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

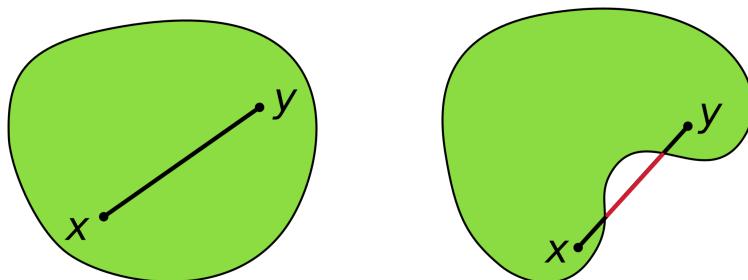
1.2.3.1 Convex Sets and Functions

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 $u, u + v$:

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

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 $B := \{v \in V : \|v\| \leq 1\}$ 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:

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

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 a 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 $C_1 + C_2 := \{c_1 + c_2 | c_1 \in C_1, c_2 \in C_2\}$ is a convex set.
2. Let C be a convex set and $\lambda \in \mathbb{R}$. The set $\lambda C := \{\lambda c | c \in C\}$ is a convex set.
3. Let C_1, \dots, 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 $f : C \rightarrow \mathbb{R}$ 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 $f : C \rightarrow \mathbb{R}$. The *epigraph* of f is the set $\text{epi}(f) := \{(u, t) \mid f(u) \leq t\}$

Claim 1.2.4 A function $f : C \rightarrow \mathbb{R}$ 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 $f : V \rightarrow \mathbb{R}$ 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 $\alpha \in [0, 1]$.

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

■

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 $f : \mathbb{R}^m \rightarrow \mathbb{R}$ be a convex function and $A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m$. The function $g : \mathbb{R}^n \rightarrow \mathbb{R}$ given by $g(u) = f(Au + b)$ is a convex function.
2. Let $f_i : V \rightarrow \mathbb{R}$ for $i = 1, \dots, m$ be a set of convex functions and $\alpha_1, \dots, \alpha_m$ positive scalars. The function $g(u) = \sum_{i=1}^m \alpha_i f_i(u)$ is a convex function.
3. Let $f_i : V \rightarrow \mathbb{R}$ for $i \in 1, \dots, m$ 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ that is defined by $f(\mathbf{w}) = (\mathbf{x}^\top \mathbf{w} - y)^2$ for some $(\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R}$. Notice that by the 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 define 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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is convex if and only if the following inequality holds for any two points $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$:

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

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

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

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 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

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

Namely, the empirical average of the parameter in question. Finally, we *predict* that we will need $T = N \times \hat{\mu}$ pizza slices. The $\hat{\mu}$ (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 $\hat{\mu}$ (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 $\hat{\mu}$. 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 $|\hat{\mu} - \mu| \leq \varepsilon$. 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 $\Omega = \{T^{n-1}H : n \in \mathbb{N}\}$. 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 (Ω, \mathcal{D}) where Ω is a *sample space* and $\mathcal{D} : 2^\Omega \rightarrow \mathbb{R}$ 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 $\mathcal{D}(A \cup B) = \mathcal{D}(A) + \mathcal{D}(B)$.

^aFor our needs this simple and intuitive definition is sufficient, though, richer definitions exist.

■ **Example 1.16** Suppose we throw two fair dice, then $\Omega = \{1, \dots, 6\}^2$ 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:

$$\begin{aligned} A &= (A \setminus B) \cup (A \cap B) \\ B &= (B \setminus A) \cup (A \cap B) \end{aligned}$$

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

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



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 (Ω, \mathcal{D}) be a probability space. The probability function is *sub-additive*, i.e., for any sequence (A_k) of events,

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

Proof. Let $B_1 = A_1$. For each $k \in \{2, 3, \dots\}$, let $B_k = A_k \setminus \cup_{i=1}^{k-1} A_i$. Note that the B_1, \dots, 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:

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

■

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 one'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 (Ω, \mathcal{D}) , a real-valued *random variable* (RV) is a function $X : \Omega \rightarrow \mathbb{R}$.

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

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

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

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

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

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

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

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

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

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

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: $\text{Var}[Z] = \mathbb{E}[Z^2] - (\mathbb{E}[Z])^2 = p - p^2 = p(1 - p)$. ■

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

Proof.

$$\begin{aligned}\mathbb{E}[X] &= \frac{1}{b-a} \int_a^b x dx = \frac{b+a}{2} \\ \mathbb{E}[X^2] &= \frac{1}{b-a} \int_a^b x^2 dx = \frac{b^2+ab+a^2}{3} \\ \text{Var}(X) &= \frac{b^2+ab+a^2}{3} - \frac{(b+a)^2}{4} = \frac{(b-a)^2}{12}\end{aligned}$$

■

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 $D(x)$ be its (unknown) distribution. Let $X_1, \dots, X_m \stackrel{i.i.d.}{\sim} D(x)$ 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, \dots, 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, $\text{Var}(X)$ is called the *population variance*.

There are many ways to estimate $\mathbb{E}[\cdot]$ and $\text{Var}(X)$ 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)

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

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

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

As mentioned before, whenever the context is clear, we will omit the subscript X and write $\hat{\mu}$ and $\hat{\sigma}^2$ instead of $\hat{\mu}_X$ and $\hat{\sigma}_X^2$. 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

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

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

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

The X_i 's are i.i.d which implies that $\mathbb{E}(x_i) = \mathbb{E}(X)$ and $\mathbb{E}(x_i^2) = \mathbb{E}(X^2)$ 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:

$$\mathbb{E}(\hat{\sigma}_X^2) = \mathbb{E}(X^2) - \mathbb{E}^2(X) = \text{Var}(X) = \sigma_X^2$$

■

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

From what we have seen in the above exercise, $\hat{\mu}_X$ is an unbiased estimator of the population mean, μ_X , and $\hat{\sigma}_X^2$ 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 .



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**: $X := (X_1, \dots, X_d)^\top$, is a finite collection of random variables, denoted X_1, \dots, X_d , defined on a common probability space (Ω, \mathcal{D}) .

Definition 1.3.13 — Joint distribution. Given random variables X_1, \dots, X_d , that are defined on a probability space, the joint probability distribution for X_1, \dots, X_d is a probability distribution that gives the probability that each of X_1, \dots, 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 $f_{X_1, X_2} : \mathbb{R}^2 \rightarrow \mathbb{R}$, such that, for any set $A \in \mathbb{R}^2$, it holds 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 $X := (X_1, \dots, X_d)^\top$, each of the scalar random variables X_1, \dots, 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, $X_1, X_2 \sim \text{Unif}([-a, a])$ 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:

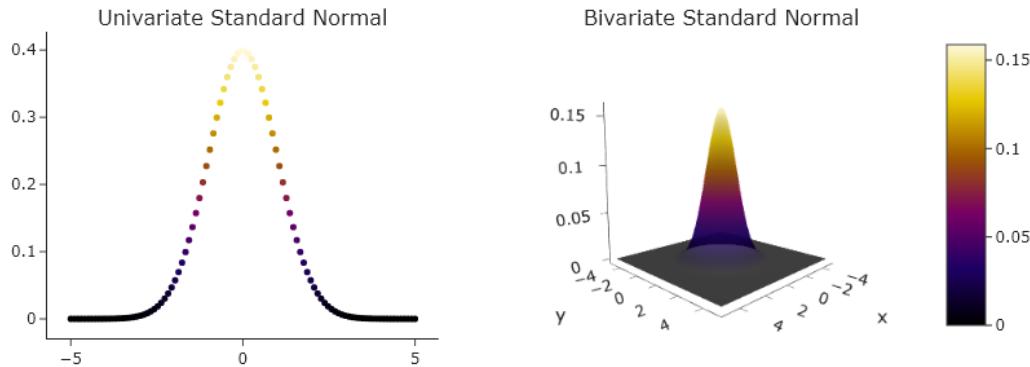


Figure 1.4: Uni- and bivariate normal distributions. [Chapter 1 Examples - Source Code](#)

Definition 1.3.17 The **Marginal distribution** of a subset of 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 $x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$. ■

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

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

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

$$\Sigma := \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\top]$$

- The diagonal elements of Σ are $\sigma(X_i, X_i) \equiv \sigma_{X_i}^2 \equiv \text{Var}(X_i)$. *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:

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

where $\hat{\mu}_i$ 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 $\hat{\Sigma}$.

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

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

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

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

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

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: $\hat{\mu} = (168, 66)^\top$, 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:

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

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

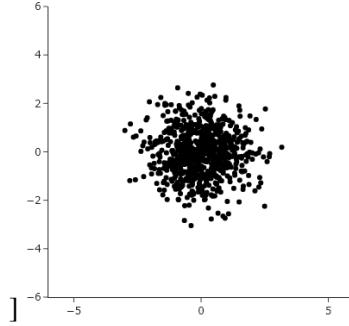


Figure 1.5: Uncorrelated variables with identical variance. [Chapter 1 Examples - Source Code](#)

variances $\sigma_{X_1}^2 = \sigma_{X_2}^2 \equiv \sigma^2$ (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 $\hat{\Sigma} = \sigma^2 I_2$. Next, will us look at how linear transformations affect our data and the sample covariance matrix $\hat{\Sigma}$ (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:

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

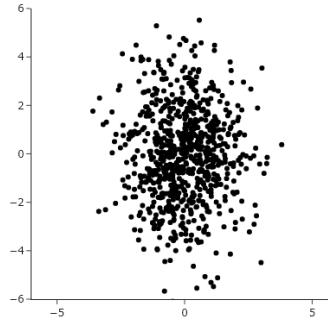


Figure 1.6: Uncorrelated scaled variables with different scaling in the x and y axis. [Chapter 1 Examples - Source Code](#)

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.

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

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.

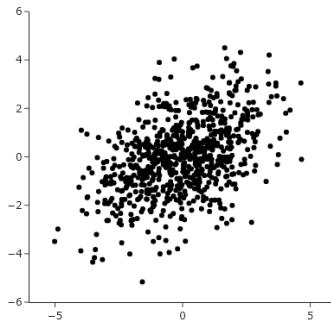


Figure 1.7: Correlated variables produced by rotation of uncorrelated variables. [Chapter 1 Examples - Source Code](#)



The form obtained above, where we multiply an orthogonal matrix R by a diagonal matrix and then by another orthogonal matrix R^T 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 (??).

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 scenarios 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")

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

where x_i equals 1 if the i 'th ball came out red and 0 otherwise. It is clear that $\mathbb{E}[\hat{p}] = p$. 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, \hat{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 \hat{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 ??.

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. $\text{Im}(X) \subseteq \mathbb{R}_{\geq 0}$) and denote 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, \dots, X_m be m i.i.d. non-negative random variables and denote the expectation value of each by $\mathbb{E}[X_i] = \mathbb{E}[X]$ $i = 1, \dots, m$. Denote also $\bar{X} = \frac{1}{m} \sum_{i=1}^m X_i$. Then for any

$a > 0$:

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

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

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

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 $\text{Var}(X)$ and for every $a > 0$

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

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

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

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

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, \dots, X_m , with a variance $\text{Var}(X_i) = \text{Var}(X)$, $i = 1..m$, we have

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

where we used the fact the $\text{Var}[X_1 + X_2] = \text{Var}(X_1) + \text{Var}(X_2)$ 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, \dots, X_m be m i.i.d random variables with a finite variance $\text{Var}(X)$. Denoting $\bar{X} = \frac{1}{m} \sum_{i=1}^m X_i$. For any $a > 0$, it holds that

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

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. Let this be a fair coin where $p = 1/2$. Let the following be a sequence of m tosses of this coin $Z_1, \dots, Z_m \stackrel{iid}{\sim} Ber(p)$ and denote the results of the tosses (that is, the samples) by $S = (z_1, \dots, z_m)$. 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 $\hat{p}(S)$ or simply \hat{p} . Since S is finite, we do not expect our estimation \hat{p} to be exact. Instead, we will settle for an algorithm that yields a \hat{p} which satisfies $|\hat{p} - p| \leq \varepsilon$, 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 $|\hat{p} - p| \leq \varepsilon$ holds with absolute certainty.

Hence, we introduce a *confidence* parameter $\delta \in (0, 1)$, and require that the event $\{S : |\hat{p} - p| > \varepsilon\}$ 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, $\hat{p}(S)$ ('inaccurate' meaning $|\hat{p} - p| > \varepsilon$) 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 $\hat{p}(S)$ that given a set of coin tosses samples $S \in \{0, 1\}^m$ returns $\hat{p} \in [0, 1]$ 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** $0 \leq p \leq 1$:

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

Namely, the probability to get a sample S such that the algorithm's output $\hat{p}(S)$ 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:

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

The function $m_{\mathcal{A}}(\varepsilon, \delta): [0, 1] \times [0, 1] \rightarrow \mathbb{N}$ 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 \hat{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 $S = (z_1, \dots, z_m)$ the most straightforward estimate of p is the empirical proportion of heads (ones), namely

$$\hat{p}(S) = \frac{1}{m} \sum_{i=1}^m z_i$$

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 $|\hat{p} - p| = 0$. Next, we will test the quality of \hat{p} as an estimator of p . In other words, how many coin flips we need to ensure that \hat{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 $|\hat{p} - p|$ as our random variable. We then need to calculate $\frac{1}{\varepsilon} \mathbb{E}[|\hat{p} - p|]$ and extract its dependence on the sample size m . By doing so we achieve the following bound:

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

If we select m to be

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

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

$$\mathcal{D}_p^m[|\hat{p}(S) - p| > \epsilon] > \delta$$

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:

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

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 $m(\epsilon, \delta) \leq \left\lceil \frac{1}{4\epsilon^2} \cdot \frac{1}{\delta} \right\rceil$.

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, \dots, X_m be independent and bounded random variables with $a_i \leq X_i \leq b_i$. Let $\bar{X} = \frac{1}{m} \sum_{i=1}^m X_i$. Then,

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

Corollary 1.3.12 Let X_1, \dots, 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 $\bar{X} = \frac{1}{m} \sum_{i=1}^m X_i$ then:

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

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

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

By placing these expressions in 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:

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

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

Corollary 1.3.13 The coin prediction algorithm, $\hat{p}(S)$, 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 $m_{\mathcal{A}}(\varepsilon, \delta) \leq \lceil \frac{1}{2\varepsilon^2} \cdot \log(\frac{2}{\delta}) \rceil$.



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 $f : \mathcal{X} \rightarrow \mathcal{Y}$ 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 $f : \mathcal{X} \rightarrow \mathcal{Y}$ 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 $i = 1 \dots m$. 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).

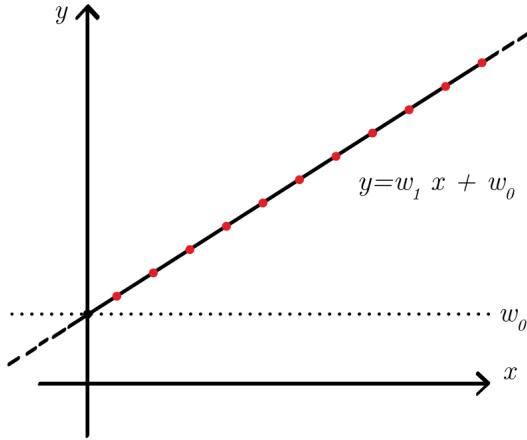


Figure 2.1: Illustration of a regression model with $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$. Red dots are samples. The solid curve is the learned prediction rule \hat{f} .

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 $\mathcal{X} \rightarrow \mathcal{Y}$ 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:

$$\mathcal{H}_{reg} := \left\{ h(x_1, \dots, x_d) = w_0 + \sum_{i=1}^d x_i w_i \mid w_0, w_1, \dots, w_d \in \mathbb{R} \right\}. \quad (2.1)$$

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, \dots, w_d representing the d

¹The 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 $\mathbf{x} = (x_1, \dots, x_d)^\top \in \mathbb{R}^d$ we add a zero-th coordinate with the value of 1, and define $\mathbf{x} = (1, x_1, \dots, x_d)^\top \in \mathbb{R}^{d+1}$. 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

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

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.

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

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} ,

$$\mathbf{X}\mathbf{w} = \mathbf{y} \quad (2.3)$$

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 Realizability

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:

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

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**

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

or the **Squared Loss**

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

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 $(\hat{f}(\mathbf{x}) - y)^2$, 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

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

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 $\hat{f}(\mathbf{x}_i) = \mathbf{x}_i^\top \mathbf{w}$ is given by:

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

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**):

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

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

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

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 $\hat{\mathbf{w}}$ such that $\mathbf{X}\hat{\mathbf{w}} = \mathbf{y}$. 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 $\hat{\mathbf{w}}$ such that $\mathbf{X}\hat{\mathbf{w}} = \mathbf{y}$. Therefore, no vector $\hat{\mathbf{w}}$ 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.

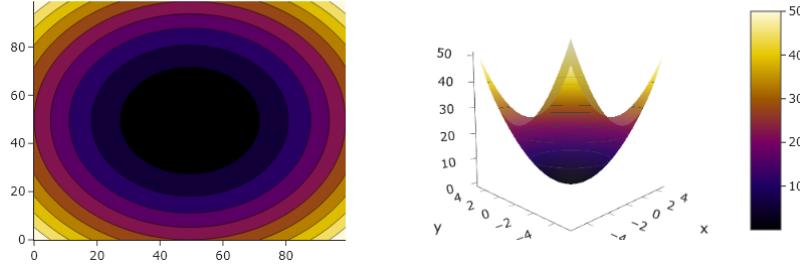


Figure 2.2: Illustration of the RSS function over \mathbb{R}^2 for \mathbf{X} of full rank. [Chapter 2 Examples - Source Code](#)

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:

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

for all $j = 0, \dots, d$, 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 ??.

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

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**:

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

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

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

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

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

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

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

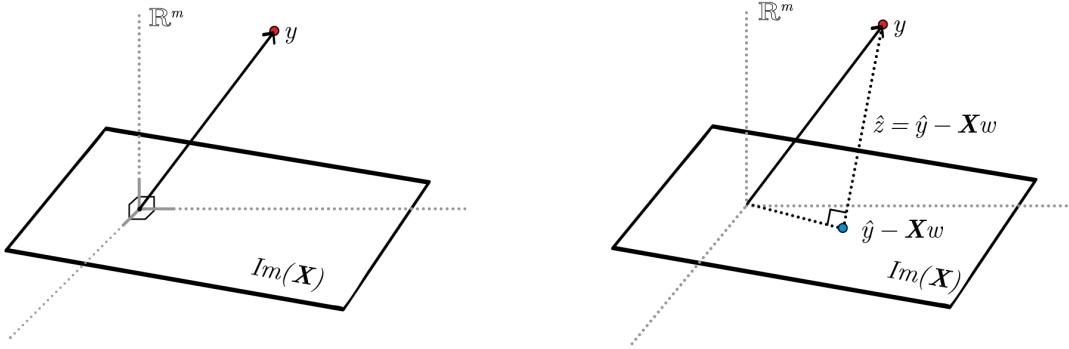
- If $\mathbf{y} \in \text{Im}(\mathbf{X})$ then by definition \mathbf{y} is a linear combination of $\varphi_0, \dots, \varphi_d$ and there exists a vector $\mathbf{w} \in \mathbb{R}^{d+1}$ such that $\mathbf{X}\mathbf{w} = \mathbf{y}$. This is the realizable case. We can now differentiate between two sub-cases:
 - If $\varphi_0, \dots, \varphi_d$ 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 $\varphi_0, \dots, \varphi_d$ 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 $\mathbf{y} \notin \text{Im}(\mathbf{X})$ then \mathbf{y} is not a linear combination of $\varphi_0, \dots, \varphi_d$. 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

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

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 $\varphi_0, \dots, \varphi_d$. Since these vectors span the subspace $\text{Im}(\mathbf{X})$, another way to write this is $\mathbf{y} - \mathbf{X}\mathbf{w} \in \text{Im}(\mathbf{X})^\perp$.

Let $\hat{\mathbf{w}}$ be a solution to the normal equations and define $\hat{\mathbf{y}} := \mathbf{X}\hat{\mathbf{w}}$. 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 $\hat{\mathbf{y}} \in \text{Im}(\mathbf{X})$. In this notation, when solving



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

(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 $\text{Im}(\mathbf{X})$. The difference $\hat{\mathbf{z}} = \mathbf{y} - \hat{\mathbf{y}}$ is therefore perpendicular (normal) to $\text{Im}(\mathbf{X})$.

Figure 2.3: Geometric interpretation of linear regression

the normal equations, namely when seeking to minimize the RSS, we minimize $\|\mathbf{y} - \hat{\mathbf{y}}\|$. Define the **residual vector** $\hat{\mathbf{z}} := \mathbf{y} - \hat{\mathbf{y}}$. Note that from (2.11) we get that $\hat{\mathbf{z}} \in \text{Im}(\mathbf{X})^\perp$. In other words, if $\hat{\mathbf{w}}$ is a solution to the normal equations then $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$ is the **orthogonal projection** of \mathbf{y} on $\text{Im}(\mathbf{X})$ and $\hat{\mathbf{z}} = \mathbf{y} - \hat{\mathbf{y}}$ is a *normal* (a perpendicular vector) to $\text{Im}(\mathbf{X})$. 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 $\text{Im}(\mathbf{X})$. 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 $\dim(\text{Ker}(\mathbf{X})) = 0$) then the projection $\hat{\mathbf{y}}$ can be described uniquely as a linear combination of the columns $\varphi_0, \dots, \varphi_d$, namely, there exists a unique $\hat{\mathbf{w}}$ such that $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$. This vector of coefficients $\hat{\mathbf{w}}$ is a unique solution to the normal equations.
 - If the columns of \mathbf{X} contain linear dependencies (equivalently, if $\dim(\text{Ker}(\mathbf{X})) > 0$) then the projection $\hat{\mathbf{y}}$ can be described as infinitely many linear combinations of the columns $\varphi_0, \dots, \varphi_d$. Any such linear combination will suffice and we simply need to find *one* vector $\hat{\mathbf{w}}$ such that $\hat{\mathbf{y}} = \mathbf{X}\hat{\mathbf{w}}$.

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

It can be shown that for any matrix A then $\text{Ker}(A) = \text{Ker}(A^\top A)$. Since we assume that $\text{Ker}(\mathbf{X})$ 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 $\mathbf{w} = [\mathbf{X}^\top \mathbf{X}]^{-1} \mathbf{X}^\top \mathbf{y}$. So in this case the unique solution to the normal equations is

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

■ **Example 2.1** Let us find the estimator $\hat{\mathbf{w}}$ for the following scenario. Suppose we are interested in estimating the running times in a 100 meters long race, based 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 $\hat{\mathbf{w}} := [\mathbf{X}^\top \mathbf{X}]^{-1} \mathbf{X}^\top \mathbf{y}$. Over given data we obtain that $\hat{\mathbf{w}} \approx (11.38, 0.003, -0.009)^\top$ (up to rounding up numbers).

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

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

■

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

The columns of \mathbf{X} are linearly dependent and therefore there are infinitely many ways 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 $\hat{\mathbf{w}}$ 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 pseudoinverse** of \mathbf{X} is defined as $\mathbf{X}^+ = V \Sigma^+ U^\top$.

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

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

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

$$A^\dagger \mathbf{b} = \operatorname{argmin}_{\mathbf{x}} \{ \|\mathbf{x}\|_2 \mid A\mathbf{x} = \mathbf{b} \} \quad (2.12)$$

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 ($\dim(\operatorname{Ker}(\mathbf{X})) = 0$) then the previously found solution $\hat{\mathbf{w}} = [\mathbf{X}^\top \mathbf{X}]^{-1} \mathbf{X}^\top \mathbf{y}$ 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.2

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 $\dim(\operatorname{Ker}(\mathbf{X})) = 0$, that the found solution is indeed a minimum.

Claim 2.1.1 Assume $\dim(\operatorname{Ker}(\mathbf{X})) = 0$ 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,

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

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

Corollary 2.1.2 The vector $\hat{\mathbf{w}} := \mathbf{X}^\dagger \mathbf{y}$ 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 $\dim(\text{Ker}(\mathbf{X})) = 0$, 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 $\hat{\mathbf{w}} = [\mathbf{X}^\top \mathbf{X}]^{-1} \mathbf{X}^\top \mathbf{y}$ 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

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

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, $\mathbf{x}_1, \dots, \mathbf{x}_m \in \mathbb{R}^d$, and m responses $y_1, \dots, y_m \in \mathbb{R}$. 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 $\hat{\mathbf{w}}$ 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 $\hat{\mathbf{w}} = [\mathbf{X}^\top \mathbf{X}]^{-1} \mathbf{X}^\top \mathbf{y}$.
- There is a nice geometric interpretation of $\hat{\mathbf{w}}$ 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 $f : \mathcal{X} \rightarrow \mathcal{Y}$ that underlies the relation $\mathcal{X} \rightarrow \mathcal{Y}$. 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 $\mathcal{X} \rightarrow \mathcal{Y}$ linear, but with an additional factor capturing randomness in the relation. Suppose now that there exists a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ 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

$$(x_i, f(\mathbf{x}_i) + z_i) \quad i = 1, \dots, m$$

with z_1, \dots, 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:

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

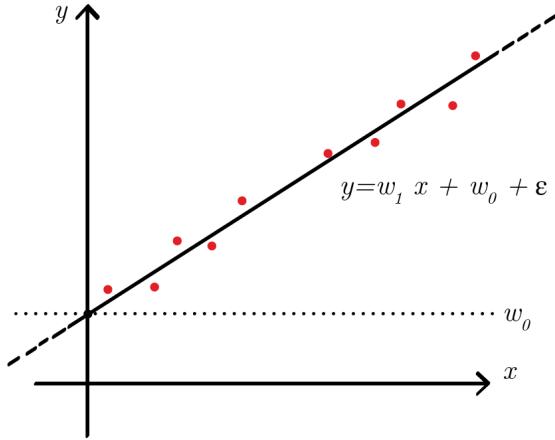


Figure 2.4: Illustration of a linear regression model where training data is noisy

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

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

Note that the vector \mathbf{y} will typically not be in $\text{Im}(\mathbf{X})$, 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 principle then:

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

This means that our learning algorithm remains the same. We learn by solving the normal equation. As mentioned, $\mathbf{y} \notin \text{Im}(\mathbf{X})$ since the noise "pushed" \mathbf{y} out of $\text{Im}(\mathbf{X})$. As we have seen, solving the normal equations is equivalent to projecting \mathbf{y} back onto $\text{Im}(\mathbf{X})$, 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 $z_i \stackrel{iid}{\sim} \mathcal{N}(0, \sigma^2)$. 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:

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

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

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

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} :

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

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

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

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

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

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 $g : \mathbb{R} \rightarrow \mathbb{R}$. We are given points $a_1 < a_2 < \dots < a_m \in \mathbb{R}$ and labels y_1, \dots, 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:

$$\mathcal{H}_{poly}^d = \left\{ x \mapsto p_{\mathbf{w}}(x) \mid \mathbf{w} \in \mathbb{R}^{d+1} \right\} \quad (2.15)$$

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

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

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

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

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 $p_{\hat{\mathbf{w}}}(a)$.



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 ??.

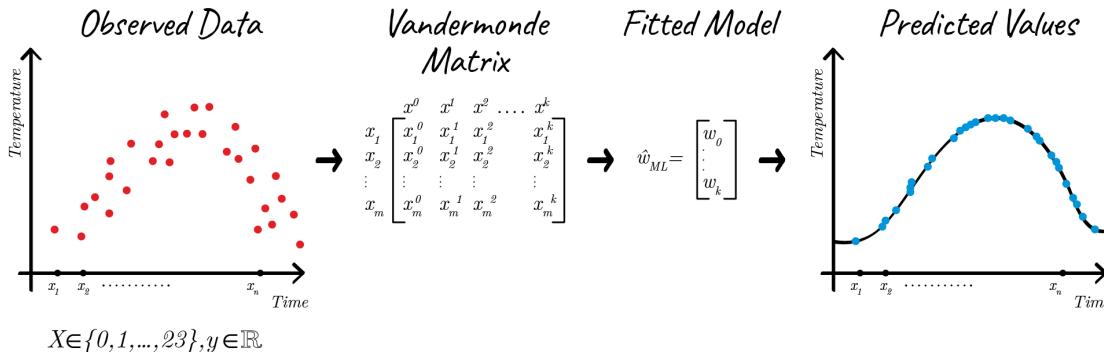


Figure 2.5: Scheme of Polynomial Fitting: Dataset of hourly temperature where x_i denotes time of day and y_i the temperature. Fitting polynomial of degree 4.

2.2.1 Bias and Variance of Estimators

Given a regression problem \mathbf{X}, \mathbf{y} we have seen how to solve $\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon}$. 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: $B(\hat{\theta}) := \mathbb{E}[\hat{\theta}] - \theta$. $\hat{\theta}$ is said to be *unbiased* if $B(\hat{\theta}) = 0$.

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: $\text{var}(\hat{\theta}) := \mathbb{E}[(\hat{\theta} - \mathbb{E}[\hat{\theta}])^2]$.

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 $S = x_1, \dots, x_m \in \mathbb{R}^d$ used to estimate some parameter: $\hat{\theta}(x_1, \dots, x_m) \stackrel{?}{\approx} \theta$. 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.

Exercise 2.1 Let \mathbf{X}, \mathbf{y} be a regression problem such that $\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon}$, $\boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 I_d)$ and $\hat{\mathbf{w}}$ being the OLS estimator. Show that $\hat{\mathbf{w}}$ is an unbiased estimator. ■

Proof.

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

where the last equality is because $\mathbb{E}[\boldsymbol{\varepsilon}] = 0$ 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

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

and let x_1, \dots, 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, \dots, x_m and generate the response value y_1, \dots, 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 $\mathbb{E}[\hat{y}] \pm 2 \cdot \text{Var}(\hat{y})$ 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 difference 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.



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

Interestingly, these two properties of bias and variance are linked. Let $\hat{\mathbf{y}} = \hat{\mathbf{y}}(S)$ denote the estimator of \mathbf{y} when using $\hat{\mathbf{w}}^{ols}$, and \mathbf{y}^* the true \mathbf{y} values. When solving the regression problem we wanted to minimize the mean square error between $\hat{\mathbf{y}}$ 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 $\bar{\mathbf{y}} = \mathbb{E}[\hat{\mathbf{y}}]$ so:

$$\begin{aligned}
 \mathbb{E}[(\hat{\mathbf{y}} - \mathbf{y}^*)^2] &= \mathbb{E}[(\hat{\mathbf{y}} - \bar{\mathbf{y}} + \bar{\mathbf{y}} - \mathbf{y}^*)^2] \\
 &= \mathbb{E}[(\hat{\mathbf{y}} - \bar{\mathbf{y}})^2] + 2(\hat{\mathbf{y}} - \bar{\mathbf{y}})\mathbb{E}[\hat{\mathbf{y}} - \bar{\mathbf{y}}] + (\bar{\mathbf{y}} - \mathbf{y}^*)^2 \\
 &= \mathbb{E}[(\hat{\mathbf{y}} - \bar{\mathbf{y}})^2] + (\bar{\mathbf{y}} - \mathbf{y}^*)^2 \\
 &= \text{var}(\hat{\mathbf{y}}) + \text{Bias}^2(\hat{\mathbf{y}})
 \end{aligned} \tag{2.18}$$

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

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

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 $\text{Ker}(\mathbf{A}) = \text{Ker}(\mathbf{A}^\top \mathbf{A})$.
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 $[\mathbf{X}^\top \mathbf{X}]^{-1} \mathbf{X}^\top = \mathbf{X}^\dagger$.



3. Classification

3.1 Classification Overview

In the previous chapter we discussed learning a regression problem where the response was 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 $\mathcal{Y} = \{1, \dots, k\}$. 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 our 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 our setup is as follows:

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

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. tobacco, 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.

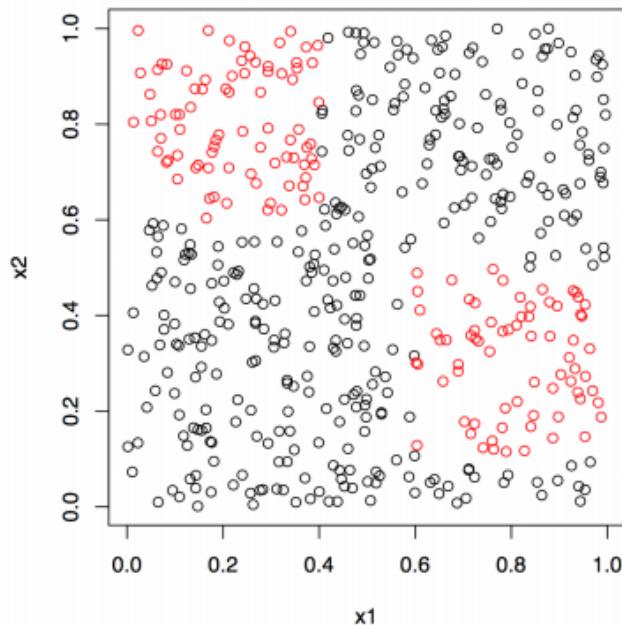


Figure 3.2: Classification training sample in \mathbb{R}^2 : Where samples are positioned in space according to the values of their features and color coded by their label.

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 $h : \mathcal{X} \rightarrow \{\pm 1\}$ and a labeled sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, the **misclassification** loss of h on this sample is:

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

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 \hat{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 $\hat{y}_i = 1$, 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 $\hat{y}_i = -1$, 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 predict 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 \hat{y}_i , the classifier recommends to give a drug which is actually potentially deadly.
- If $y_i = 1$ and \hat{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 concerned 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: $(TP + TN) / (P + N) = 1 - \text{Error Rate}$.
- 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 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:

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

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 classifiers, 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.

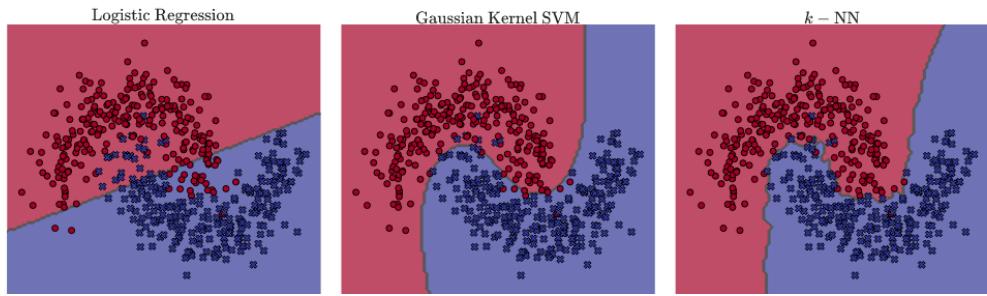


Figure 3.3: Decision Boundaries of classifiers fitted over moons dataset. [Chapter 3 Examples - Source Code](#)

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 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 $\mathcal{Y} := \{\pm 1\}$.

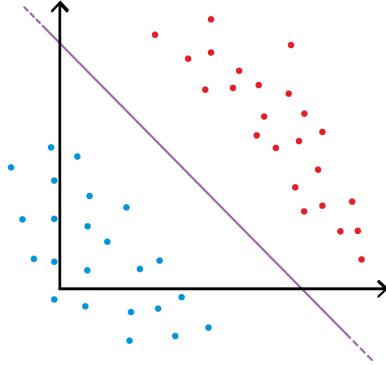


Figure 3.4: Half-space Classification Illustration: For a domain-set $\mathcal{X} \in \mathbb{R}^2$ the two classes, coded as red and blue colors, are linearly separable.

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

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

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

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

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

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:

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

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:

$$\mathbb{R}^d = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^\top \mathbf{w} > 0 \right\} \uplus \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^\top \mathbf{w} = 0 \right\} \uplus \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^\top \mathbf{w} < 0 \right\} \quad (3.4)$$

These sets correspond to the open half spaces on either side of the hyperplane $\mathbf{w}^\perp = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}^\top \mathbf{w} = 0 \right\}$ 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.

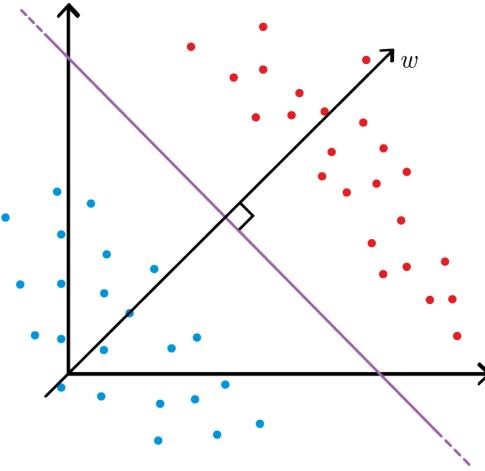


Figure 3.5: Corresponding Hyperplane to \mathbf{w}^\perp

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

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

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

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

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-homogeneous 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:

$$\mathcal{H}_{half} := \left\{ h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{x}^\top \mathbf{w}) \mid \mathbf{w} \in \mathbb{R}^d \right\} \quad (3.5)$$

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 mis-classified training samples are exactly those where $y_i \cdot \text{sign}(\mathbf{x}^\top \mathbf{w}) = -1$ or equivalently $y_i \cdot \mathbf{x}^\top \mathbf{w} < 0$. So defining the loss of a given hypothesis over S is:

$$L_S(h_{\mathbf{w}}) := \sum_{i=1}^m \mathbb{1}[y_i \cdot \mathbf{x}^\top \mathbf{w} < 0] \quad (3.6)$$

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 $L_S(h_{\mathbf{w}}) = 0$. 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 $L_S(h_{\mathbf{w}})$.

3.2.2 Computational Implementation

Once we realize what learning principle we want to apply, we need to find a computationally efficient algorithm to find the desired hypothesis. As we are applying the ERM principle we would like to efficiently compute

$$\hat{\mathbf{w}} := \underset{\mathbf{w} \in \mathbb{R}^d}{\operatorname{argmin}} L_S(h_{\mathbf{w}}) \quad (3.7)$$

By the realizability assumption, there exists a vector $\mathbf{w}_0 \in \mathbb{R}^d$ such that $y_i \cdot \mathbf{x}^\top \mathbf{w}_0 > 0 \quad i = 1, \dots, m$. This implies that there also exists a (different) vector $\mathbf{w}_1 \in \mathbb{R}^d$ such that $y_i \cdot \mathbf{x}^\top \mathbf{w}_1 \geq 1 \quad i = 1, \dots, m$. That is because we can simply normalize \mathbf{w}_0 by the smallest product:

$$\mathbf{w}_1 := \frac{1}{\min_i \{y_i \cdot \mathbf{x}^\top \mathbf{w}_0\}} \cdot \mathbf{w}_0$$

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

$$y_i \cdot \text{sign}(\mathbf{x}^\top \mathbf{w}) \geq 1 \quad \forall i = 1, \dots, m \quad (3.8)$$

3.2.3 Convex Optimization

In (3.8) we therefore understand that the problem of finding a linear separator is in fact a problem of finding a vector that satisfies m linear constraints. As these constraints are all convex constraints this is an example 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:

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} && f_0(\mathbf{x}) \\ &\text{subject to} && f_i(\mathbf{x}) \leq b_i \quad i = 1, \dots, n \end{aligned}$$

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

Then, naturally, a convex optimization problem is an optimization problem as above in which f_0, f_1, \dots, 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:

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

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.

Definition 3.2.5 — Quadratic Program. An optimization problem is called a Quadratic Program (QP) if it can be written in the following form:

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

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.

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.

3.2.4 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:

$$\begin{array}{ll} \text{minimize} & 0 \\ \text{subject to} & y_i \cdot \mathbf{x}^\top \mathbf{w} \geq 1 \quad i = 1, \dots, m \end{array} \tag{3.9}$$

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.

3.2.4.1 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 $\mathbf{w}^{(0)}, \mathbf{w}^{(1)}, \dots$, where each vector is derived from the vector preceding it. At each iteration t we search for a sample i which is misclassified by $\mathbf{w}^{(t)}$. Then, we update $\mathbf{w}^{(t)}$ by moving it in the direction of the misclassified sample $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$.

Algorithm 1 Batch-Perceptron

```

procedure PERCEPTRON( $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ )
     $\mathbf{w}^{(0)} \leftarrow 0$                                       $\triangleright$  Initialize parameters
    for  $t = 1, 2, \dots$  do
        if  $\exists i$  s.t.  $y_i \langle \mathbf{w}^{(t)}, \mathbf{x}_i \rangle \leq 0$  then
             $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + y_i \mathbf{x}_i$ 
        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 $y_i \cdot \mathbf{x}_i^\top \mathbf{w} > 0 \quad i = 1, \dots, m$. 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.



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 in 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 ??.

3.2.5 Learner ID Card

- **Hypothesis class:** the class of linear separators (3.5)
- **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

Figure 3.6: Perceptron Fitting: Fit a separating hyperplane using Perceptron algorithm. [Chapter 3 Examples - Source Code](#)

- **Prediction of new sample:** $\hat{y}_{new} := \text{sign}(\mathbf{x}_{new}^\top \mathbf{w} + b)$
- **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 (3.9) 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 existence 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-realizable case). In this case the optimization problem described in [3.9](#) is computationally hard.

Returning to the hypothesis class of non-homogeneous separating half-spaces \mathcal{H}_{half} (3.3), we would like to ease the realizability assumption. 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

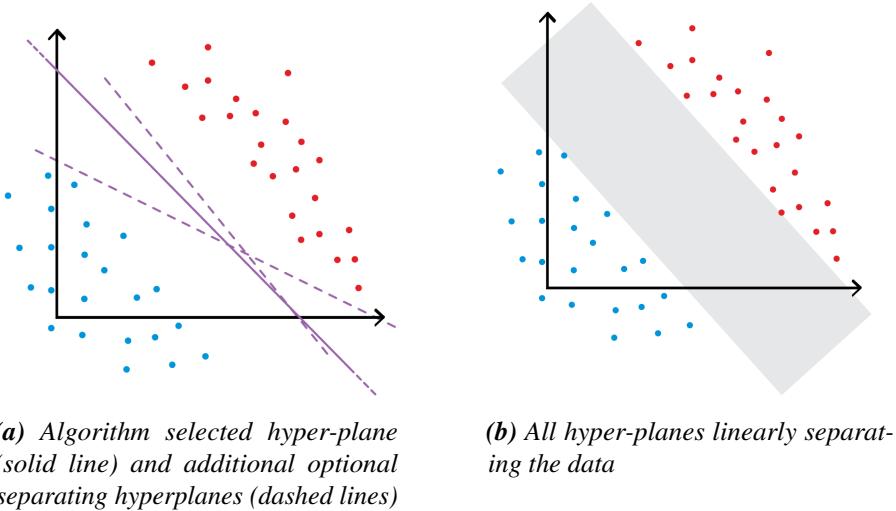


Figure 3.7: Illustration of the existance of multiple separating hyper-planes

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:

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

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

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

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.

3.3.2 Hard-SVM

Let's 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:

$$\begin{aligned} & \text{maximize} && M((\mathbf{w}, b), S) \\ & \text{subject to} && y_i \cdot (\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 \quad i = 1, \dots, m \end{aligned} \tag{3.10}$$

The optimization variables are $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$. Comparing with the linear program of half-spaces (3.9) we see that the constraints are kept, which ensure the hyperplane chosen separates the training

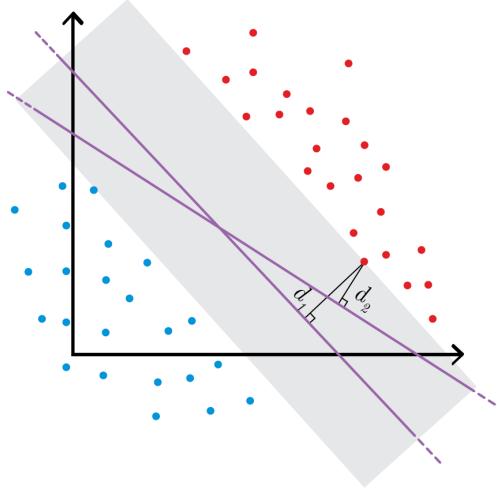


Figure 3.8: Margin of specified hyper-planes

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 (3.10), 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 ourselves 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:

$$\inf_{\mathbf{v} \in B} \|\mathbf{x} - \mathbf{v}\|^2$$

Exercise 3.1 Let $(\mathbf{w}, b) \in \mathbb{R}^d \times \mathbb{R}$ be a hyperplane where $\|\mathbf{w}\| = 1$ and $\mathbf{x} \in \mathbb{R}^d$ then the distance between \mathbf{x} and the hyperplane (\mathbf{w}, b) is $|\langle \mathbf{w}, \mathbf{x} \rangle + b|$. ■

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

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

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

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

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} :

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

■

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:

$$\begin{array}{ll}
 \underset{\|\mathbf{w}\|=1, b}{\operatorname{argmax}} & \min_{1 \leq i \leq m} |\langle \mathbf{w}, \mathbf{x}_i \rangle + b| \\
 \text{subject to} & y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 \quad i = 1, \dots, m
 \end{array} \tag{3.11}$$

While the constraints enforce \mathbf{w} to define a separating hyperplane, the objective will make us choose a separating hyperplane with the maximal margin. To solve this problem numerically, we will need to perform a slight manipulation and write it in a different form.

Claim 3.3.1 Let (\mathbf{v}^*, c^*) be an optimal solution of:

$$\begin{array}{ll}
 \underset{(\mathbf{w}, b)}{\operatorname{argmin}} & \|\mathbf{w}\|^2 \\
 \text{subject to} & y_i \cdot (\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 \quad i = 1, \dots, m
 \end{array} \tag{3.12}$$

Then, $\mathbf{w}^* := \gamma \mathbf{v}^*, b^* := \gamma c^*$ for $\gamma = \|\mathbf{v}^*\|^{-1}$ is an optimal solution for 3.3.2.1.

Proof. Let us begin with simplifying (3.3.2.1). Consider a *feasible* solution \mathbf{w} to the problem (i.e. that satisfies all constraints). It holds that $|\langle \mathbf{w}, \mathbf{x}_i \rangle + b| = y_i (\mathbf{x}_i^\top \mathbf{w} + b)$. Hence, we can rewrite (3.3.2.1) as:

$$\begin{array}{ll}
 \underset{\|\mathbf{w}\|=1, b}{\operatorname{argmax}} & \min_{1 \leq i \leq m} y_i (\mathbf{x}_i^\top \mathbf{w} + b) \\
 \text{subject to} & y_i \cdot (\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 \quad i = 1, \dots, m
 \end{array}$$

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

$$\underset{\|\mathbf{w}\|=1, b}{\operatorname{argmax}} \min_{1 \leq i \leq m} y_i (\mathbf{x}_i^\top \mathbf{w} + b)$$

Next, let us examine the optimization in claim. Notice that an optimal solution \mathbf{v}^*, c^* will always satisfy:

$$\min_{1 \leq i \leq m} y_i (\langle \mathbf{v}^*, \mathbf{x}_i \rangle + c^*) = 1$$

since otherwise, we can divide \mathbf{v}^* by a positive number and get a feasible solution with a lower objective. As scalar multiplication does not change the hyperplane denote $\gamma = \|\mathbf{v}^*\|^{-1}$ and let

$\mathbf{w}^* := \gamma \mathbf{v}^*, b^* := \gamma c^*$. Since (\mathbf{v}^*, c^*) is a is the optimal solution of (3.12) it must satisfy all constraints and is therefore a feasible solution for (*) with an objective:

$$\min_i y_i (\langle \mathbf{w}^*, \mathbf{x}_i \rangle + b^*) = \min_i y_i (\langle \gamma \mathbf{v}^*, \mathbf{x}_i \rangle + \gamma c^*) = \gamma$$

Assume towards contradiction that there is a solution $(\mathbf{w}_2, b_2) \neq (\mathbf{w}^*, b^*)$ achieving a higher objective:

$$\min_i y_i (\langle \gamma \mathbf{w}_2^*, \mathbf{x}_i \rangle + \gamma b_2^*) = \delta > \gamma$$

But then $(\mathbf{w}_2/\delta, b_2/\delta)$ is a feasible solution for (3.12) which $\|\mathbf{w}_2/\delta\| = \delta^{-1} < \gamma^{-1}$ contradicting optimality of (\mathbf{v}^*, c^*) . ■

This means in fact that maximizing the margin is equivalent to minimizing the size of the hyperplane. The optimization problem written in (3.12) is a **Quadratic Program (QP)** for which there exist efficient solves. By using them to solve problem (3.12) we can obtain an optimal solution for the Hard-SVM optimization problem.



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 $y_i \cdot (\mathbf{x}_i^\top \mathbf{w} + b) \geq 1$ 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 $y_i \cdot (\mathbf{x}_i^\top \mathbf{w} + b) < 0$ then sample \mathbf{x}_i is on the “wrong side“ of the hyperplane. This means that:

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

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:

$$\begin{aligned} & \text{minimize}_{\mathbf{w}} \quad \|\mathbf{w}\|^2 \\ & \text{subject to} \quad \begin{cases} y_i \cdot (\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 - \xi_i & i = 1, \dots, m \\ \xi_i \geq 0 \quad \wedge \quad \frac{1}{m} \sum_{i=1}^m \xi_i \leq C \end{cases} \end{aligned} \quad (3.13)$$

where $C > 0$ is a constant we specify. The variables ξ_1, \dots, ξ_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“.

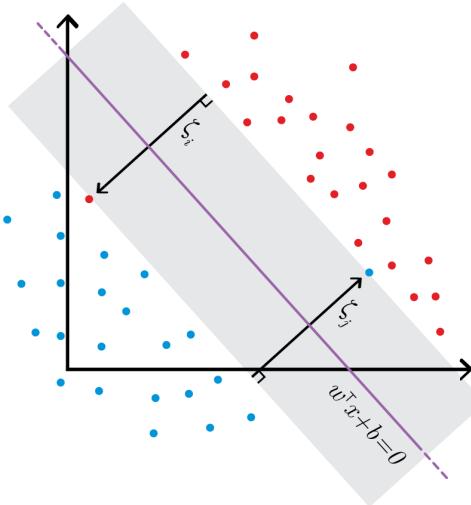


Figure 3.9: Slack variables of data-points that are on the “wrong” side of the hyper-plane.

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

$$\begin{aligned} & \underset{\mathbf{w}, \{\xi_i\}}{\text{minimize}} \quad \lambda \|\mathbf{w}\|^2 + \frac{1}{m} \sum_{i=1}^m \xi_i \\ & \text{subject to} \quad y_i \cdot (\mathbf{x}_i^\top \mathbf{w} + b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \quad i = 1, \dots, m \end{aligned} \quad (3.14)$$

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

$$\ell^{\text{hinge}}(a) = \max \{0, 1 - a\}, \quad a \in \mathbb{R} \quad (3.15)$$

Claim 3.3.2 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

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

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

Proof. Given a specific hyperplane (\mathbf{w}, b) consider the minimization over ξ_1, \dots, ξ_m . Since we defined the auxiliary variables to be nonnegative, the optimal assignment of ξ_i is

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

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

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 (3.14) (or equivalently C in (3.13)), 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 ??.

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:** $\hat{y}_{\text{new}} := \text{sign}(\mathbf{x}_{\text{new}}^\top \mathbf{w} + b)$
- **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 (??)

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 $\mathcal{X} \rightarrow \mathcal{Y}$ as $\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon$ and $\varepsilon \sim \mathcal{N}(0, \sigma^2 I_m)$. Notice that as ε is a random variable, \mathbf{y} too is a random variable distributing as a multi-variate Gaussian:

$$\mathbf{y} \sim \mathcal{N}(\mathbf{X}\mathbf{w}, \sigma^2 I_m) \quad (3.16)$$

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 :

$$p(y_i | \mathbf{x}_i, \mathbf{w}) = \mathcal{N}(y_i | \phi_{\mathbf{w}}(\mathbf{x}_i), \sigma^2) \quad \text{where} \quad \phi_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}^\top \mathbf{w} \quad (3.17)$$

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:

$$p(y_i|\mathbf{x}_i) = \text{Ber}(y_i|p(\mathbf{x}_i)) \quad (3.18)$$

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 $\phi_{\mathbf{w}}$ is \mathbb{R} while $p(\mathbf{x}_i)$ is restricted to $[0, 1]$. Instead, we would like to choose some **link** function $\phi_{\mathbf{w}} : \mathbb{R} \rightarrow [0, 1]$ that is monotone increasing and maps $(-\infty, \infty)$ bijectively to $(0, 1)$. Define the relation to be:

$$p(y_i|\mathbf{x}_i, \mathbf{w}) = \text{Ber}(y_i|\phi_{\mathbf{w}}(\mathbf{x}_i)), \quad \phi_{\mathbf{w}} := \text{sigm}(\mathbf{x}^\top \mathbf{w}) \quad (3.19)$$

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

$$\text{sigm}(\mathbf{a}) := \frac{e^{\mathbf{a}}}{e^{\mathbf{a}} + 1} \quad (3.20)$$

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

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

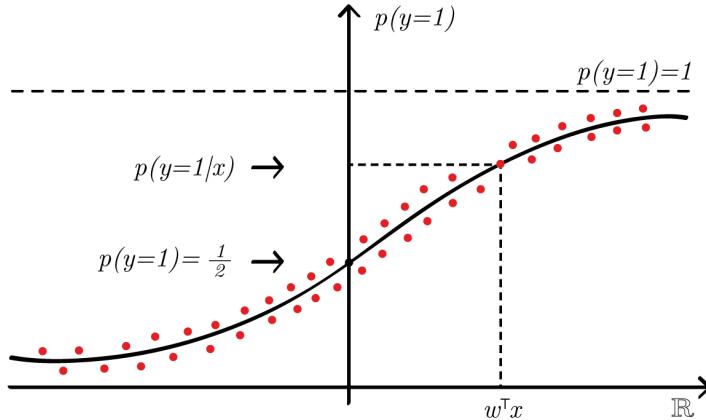


Figure 3.10: Illustration of fitted logit function for values corresponding to $\mathbf{x}^\top \mathbf{w}$.

- R** In (3.19) 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 $\text{Multinomial}(p_1, \dots, p_K)$, $\sum_p i = 1, 0 \leq p_i \leq 1$. 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:

$$\mathcal{H}_{\text{logistic}} := \left\{ h_{\mathbf{w}}(\mathbf{x}) = \text{sigm}(\mathbf{x}^\top \mathbf{w}) \mid \mathbf{w} \in \mathbb{R}^{d+1} \right\} \quad (3.21)$$

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 $h_{\mathbf{w}} : \mathbb{R}^{d+1} \rightarrow [0, 1]$ and not $h_{\mathbf{w}} : \mathbb{R}^{d+1} \rightarrow \{0, 1\}$ as required for classification problems. Since $\{0, 1\} \subset [0, 1]$, we can use the training sample to select a function in $\mathcal{H}_{\text{logistic}}$. This means we will be able to train a model, but how will we predict over new samples? Suppose our learner chose some $h_{\mathbf{w}} \in \mathcal{H}_{\text{logistic}}$. 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: $\hat{y} := \mathbb{1}[h_{\mathbf{w}}(\mathbf{x}) > \alpha]$. 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 (3.19) and hypothesis class (3.21), 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 $y_i \sim \text{Ber}(\phi_{\mathbf{w}}(\mathbf{x}))$ where ϕ is the logistic function. Therefore, the likelihood of $\mathbf{w} \in \mathbb{R}^{d+1}$ is:

$$\begin{aligned} \mathcal{L}(\mathbf{w} | \mathbf{X}, \mathbf{y}) &= \mathbb{P}(y_1, \dots, y_m | \mathbf{X}, \mathbf{w}) \\ &= \prod \mathbb{P}(y_i | \mathbf{x}_i, \mathbf{w}) \\ &= \prod_{i:y_i=1} \mathbb{P}(y_i | \mathbf{x}_i, \mathbf{w}) \cdot \prod_{i:y_i=0} \mathbb{P}(y_i | \mathbf{x}_i, \mathbf{w}) \\ &= \prod_{i:y_i=1} p_i(\mathbf{w}) \cdot \prod_{i:y_i=0} (1 - p_i(\mathbf{w})) \\ &= \prod p_i(\mathbf{w})^{y_i} (1 - p_i(\mathbf{w}))^{1-y_i} \end{aligned} \quad (3.22)$$

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

$$\begin{aligned} \ell(\mathbf{w} | \mathbf{X}, \mathbf{y}) &= \sum_{i=1}^m [y_i \log(p_i(\mathbf{w})) + (1 - y_i) \log(1 - p_i(\mathbf{w}))] \\ &= \sum_{i=1}^m \left[y_i \log \left(\frac{e^{\mathbf{x}_i^\top \mathbf{w}}}{1 + e^{\mathbf{x}_i^\top \mathbf{w}}} \right) + (1 - y_i) \log \left(\frac{1}{1 + e^{\mathbf{x}_i^\top \mathbf{w}}} \right) \right] \\ &= \sum_{i=1}^m \left[y_i \cdot \mathbf{x}_i^\top \mathbf{w} - \log \left(1 + e^{\mathbf{x}_i^\top \mathbf{w}} \right) \right] \end{aligned} \quad (3.23)$$

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

$$\hat{\mathbf{w}} := \underset{\mathbf{w} \in \mathbb{R}^{d+1}}{\operatorname{argmax}} \sum_{i=1}^m \left[y_i \cdot \mathbf{w}^\top \mathbf{x}_i - \log \left(1 + e^{\mathbf{w}^\top \mathbf{x}_i} \right) \right] \quad (3.24)$$

(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 (3.24) 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 property 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

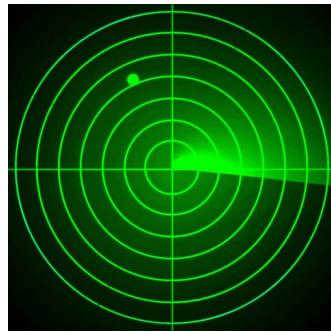
As we will encounter later in this chapter, in many classification scenarios we face the following question: Suppose we trained some classifier $h \in \mathcal{H}$ and derive classifications by the following rule: for some cutoff value $\alpha \in [0, 1]$

$$\hat{y} := \begin{cases} 1 & h(\mathbf{x}) > \alpha \\ 0 & h(\mathbf{x}) \leq \alpha \end{cases}$$

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 **more** likely to give 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.

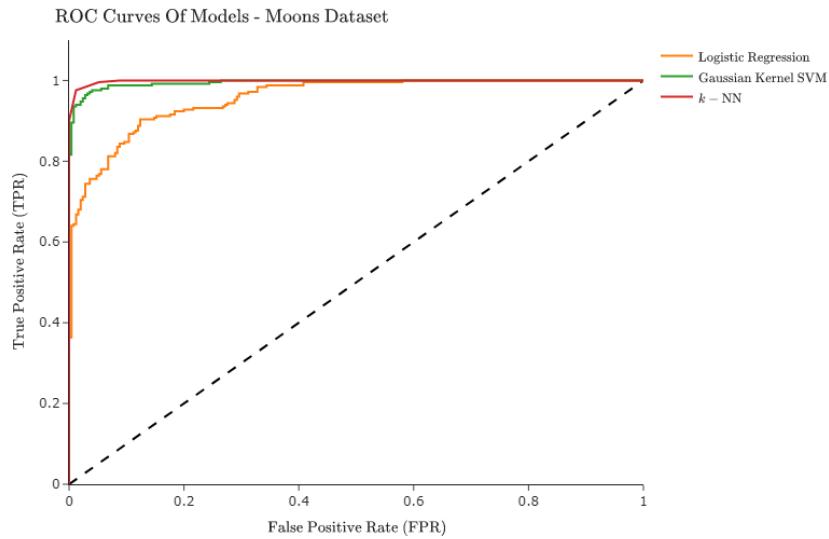


Figure 3.11: ROC Curve of classifiers fitted over moons dataset. [Chapter 3 Examples - Source Code](#)

- **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 (3.21)
- **Learning principle used for training:** Maximum likelihood
- **Computational implementation:** Specialized iterative method based on Newton-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 interpret which features drive the classification as well as understand why a given sample was predicted as it was

- **Family of models:** As seen in ?? 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 $\hat{y}_{new} := \mathbb{1}[\text{sigm}(\mathbf{x}_{new}^\top \mathbf{w} + b) \geq \alpha]$
- **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:

$$\mathcal{H}_{logistic} = \left\{ \mathbf{x} \mapsto \text{sigm}(\mathbf{x}^\top \mathbf{w}) \right\}$$

Then, we have derived an optimization problem using the maximum likelihood principle (3.24). 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 $\hat{y} := \mathbb{1}[\text{sigm}(\mathbf{x}_{new}^\top \mathbf{w}) \geq \alpha]$.

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.



This family of learners 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 $\rho : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}_+$. 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}') \triangleright 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 $\pi = (\pi_1, \dots, \pi_m)$ the permutation of $(1, \dots, m)$ such that $d_{\mathbf{x}_{\pi_1}} \leq \dots \leq d_{\mathbf{x}_{\pi_m}}$

 Select k nearest samples $\mathbf{x}_{\pi_1}, \dots, \mathbf{x}_{\pi_k}$ and predict by majority vote:

$$\hat{y} := \underset{y \in \{0,1\}}{\operatorname{argmax}} \sum_{i=1}^k \mathbb{1}[y_{\pi_i} = y]$$

return \hat{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 $k \approx \dots$ 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 a 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} into a disjoint union of axis-parallel boxes, and to assign a class prediction to each box.

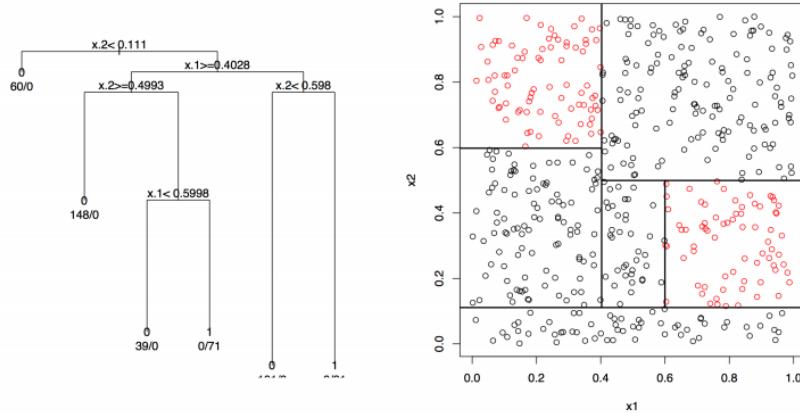


Figure 3.13: Decision tree and induced partitioning of \mathbb{R}^2 sample space

To make our 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 \mathbb{R}^d into two axis-parallel "boxes" (half-spaces). We obtain:

$$B_1^+ = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}_{i_1} > t_1 \right\}, \quad B_1^- = \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}_{i_1} \leq t_1 \right\}$$

- 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.

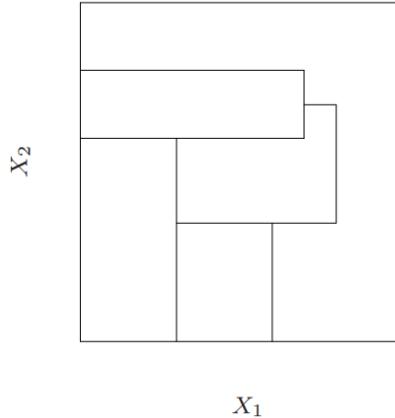


Figure 3.14: Partitioning \mathbb{R}^2 into axis-aligned boxes not describing a tree partition

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 $\mathbb{R}^d = \bigcup_{j=1}^N B_j$ of \mathbb{R}^d into N boxes, and label assignments $c_j \in \{0, 1\}$ ($j = 1, \dots, N$) assigning label c_j to box B_j , the hypothesis $h \in \mathcal{H}_{CT}$ is a function $h : \mathbb{R}^d \rightarrow \{0, 1\}$ defined by

$$h(\mathbf{x}) := \sum_{j=1}^N c_j \mathbb{1}[\mathbf{x} \in B_j]$$

■ **Example 3.4** Consider the following scenario: suppose someone comes into a hospital 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 **textbf{emergency}**
 - 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 of course what learning principle shall we use. That is, how shall we 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 in some manner, that consists of N disjoint boxes $\mathbb{R}^d = \bigcup_{j=1}^N B_j$. Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be our training 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

$$\sum_{\mathbf{x}_i \in B_j} \mathbb{1}[y_i \neq \hat{y}(B_j)] \quad (3.25)$$

Let us begin learning by applying the ERM principle. Denote the fraction of correctly classified samples with label $y \in \{0, 1\}$ in some box B by:

$$P_y^S(B) := \frac{1}{n_S(B)} \sum_{\mathbf{x}_i \in B} \mathbb{1}[y_i = y] \quad (3.26)$$

where $n_S(B)$ is the number of training samples that fall inside the box B . 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

$$\hat{y}_S(B) := \operatorname{argmax}_{y \in \{0, 1\}} P_y^S(B) \quad (3.27)$$

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

$$\operatorname{argmin}_{h \in \mathcal{H}_{CT}} L_S(h) \quad (3.28)$$

where $L_S(h)$ is the misclassification error of h over S .

Looking back at the described procedure could you 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

$$\operatorname{argmin}_{h \in \mathcal{H}_{CT}^k} L_S(h)$$

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 $\operatorname{argmin}_{h \in \mathcal{H}_{CT}^k} L_S(h)$ 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 a closed 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}_{CT}^k 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}_{CT}^k is an NP-Hard problem with respect to the training sample size¹.

This is our first encounter with the bitter truth that though the ERM principle is nice, it is often

¹By reduction from "three dimensional matching", see Hyafil and Rivest, "Constructing Optimal Binary Decision Trees is NP-Complete", Information Processing Letters 5(1), 1976

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 consists 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, let B be some box in an existing disjoint partitioning of \mathbb{R}^d (at the very beginning this box is the entire sample space). Splitting B is done as follows:

- For each coordinate $i \in [d]$ and each potential chopping value $t \in \mathbb{R}$ we define $g_i(t)$ by:
 - For each $t \in \mathbb{R}$ let us define the two boxes obtained from B by splitting along coordinate i at value t :

$$B_t^+ := \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}_i > t \right\}, \quad B_t^- := \left\{ \mathbf{x} \in \mathbb{R}^d \mid \mathbf{x}_i \leq t \right\}$$

- Denote $\hat{y}_S(B_t^\pm)$ be the class assignment for boxes B_t^\pm as defined above, minimizing the empirical misclassification risk in that box.
- Denote $P_{\hat{y}_S(B_t^\pm)}^S(B_t^\pm)$ be the optimal empirical misclassification risk incurred by these class assignments.
- Now, let $g_i(t)$ be the best empirical risk incurred by chopping box B at value t along coordinate i :

$$g_i(t) := P_{\hat{y}_S(B_t^-)}^S(B_t^-) + P_{\hat{y}_S(B_t^+)}^S(B_t^+)$$

- Denote t_i to be the **best** chopping point along coordinate i and calculate it for every coordinate:

$$t_i := \underset{t \in \mathbb{R}}{\operatorname{argmin}} g_i(t) \quad i = 1, \dots, d$$

- Let i^* denote the **best** coordinate along which to chop B :

$$i^* := \underset{i \in [d]}{\operatorname{argmin}} g_i(t_i)$$

- Chop B along coordinate i^* at value t_{i^*} .

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 exponential 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 $\mathcal{O}(m^2d)$

3.6.5 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 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 two leaves together and removing the node between them. Hence, “pruning”. We will complete the CART heuristic when we discuss regularization (??).

3.6.6 Interpretability

One of the great advantages of a classification tree is that it is very interpretable. To understand which features were 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.

3.6.7 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 ??