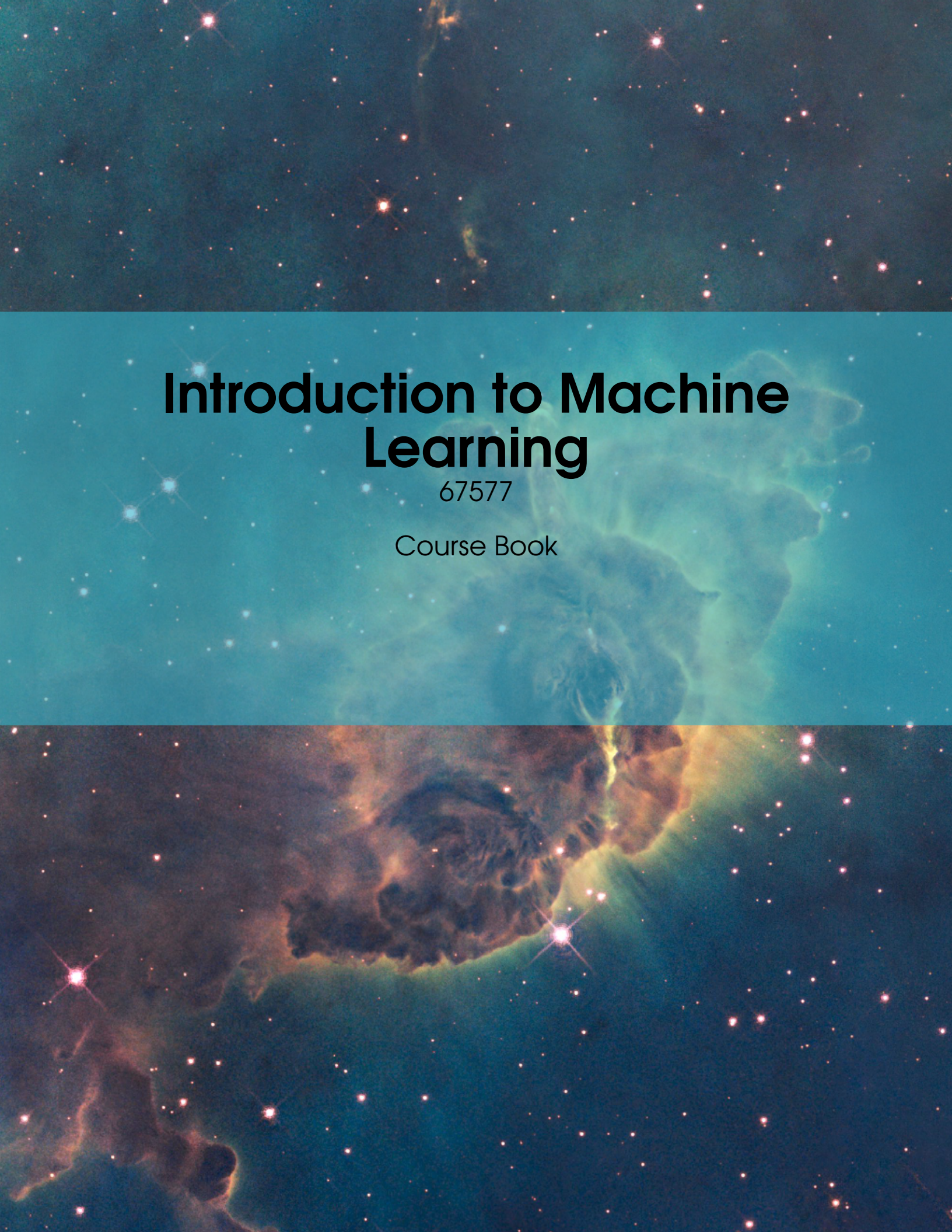


Introduction to Machine Learning

67577

Course Book

Introduction to Machine Learning (67577) - Course Book

The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University, Jerusalem

Written by Gilad Green, Ury add surname and Prof. Matan Gavish

First release, March 2021

Contents

0.1 Preface

0.1.1 Notation

Table 1: *Notation Summary*

Symbol	Meaning
$\mathbb{R}$	The set of real numbers.
$\mathbb{R}^d$	The set of d-dimensional vectors over the field of real numbers.
$\mathbb{R}_+$	The set of non-negative real numbers.
$\mathbb{N}$	The set of natural numbers.
$\{0,\ldots,m\}$	The set of natural numbers between 0 and m.
$[m]$	The set of natural numbers between 1 and m.
$\mathbf{x},\mathbf{y},\mathbf{w}$	Column vectors.
x_i, y_i, w_i	The i-th coordinate of a vector.
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
$S = x_1,\ldots,x_n$	A sequence of samples from domain set
$S = z_1,\ldots,z_n$	A sequence of samples and corresponding responses from product space $\mathcal{Z}$

0.1.2 Data Sets Used in Book, Labs and Examples

1. Mathematical Basis

1.1 Linear Algebra

1.1.1 Linear Transformations

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

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

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

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

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

Let us define some vector spaces associated with each linear transformation

Definition 1.1.3 Let A be the matrix corresponding the linear transformation $T : V \to W$. We define the:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

■

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

1.1.2 Norms, Inner Products and Projections

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

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

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

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

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

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

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

■

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

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

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

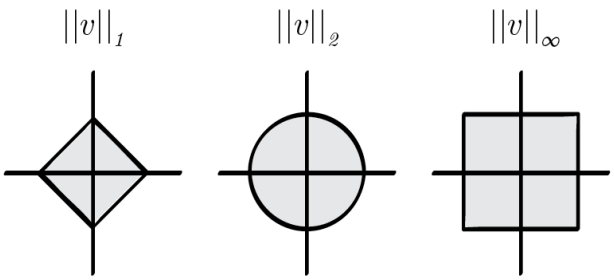
- Triangle inequality: $||v+u|| \leq ||v|| + ||u||$.

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

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

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

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



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

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

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

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

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

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

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

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

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

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

On the other hand we also know that:

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

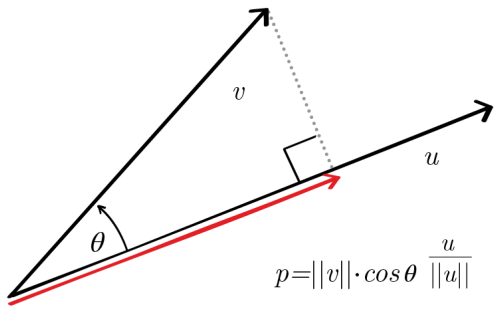
Hence, we conclude that:

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

■

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

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



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

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

Definition 1.1.11 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 1.1.12 Let V be a k-dimensional subspace of $\mathbb{R}^d$, and let $v_1,\ldots,v_k$ be an orthonormal basis of V. Define $P = \sum_{i=1}^k v_i v_i^\top$. The matrix P is an ***orthogonal projection matrix*** onto the subspace V.

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

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

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

R Notice that the definition of the projection matrix includes a sum of outer products

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 1.1.13 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 1.1.14 — 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 $Av = \lambda v$.

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

Theorem 1.1.6 — EVD. Let $A \in \mathbb{R}^{d\times d}$ be a real symmetric matrix. Then there exist an orthonormal

matrix $U \in \mathbb{R}^{d\times d}$ and a diagonal matrix D such that, $D_{i,i}$, $i = 1..n$ are the eigenvalues of A and $A = UDU^\top$.

This decomposition of A is called Eigenvalues Decomposition (EVD). It is widely used and has some strong properties. 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 very easy to compute the inverse of A, if it exists: $A^{-1} = UD^{-1}U^\top$.

A drawback of the EVD is the restriction to square symmetric matrices. Though this is a rich family of matrices we would like to derive some useful decomposition for non-symmetric and even non-square matrices.

Definition 1.1.15 Let $(V, ||\cdot||)$ be a normed space. We say that $v \in V$ is a unit vector iff $||v|| = 1$.

Definition 1.1.16 Let $A \in \mathbb{R}^{m\times d}$ and let $v \in \mathbb{R}^d$, $u \in \mathbb{R}^d$ 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 Decoposition (SVD). Let $A \in \mathbb{R}^{m\times d}$ be a real matrix. A can be written as a singular value decomposition of the form $A = U\Sigma V^\top$, where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{d\times d}$ are orthonormal matrices, and $\Sigma \in \mathbb{R}^{m\times d}$ is a diagonal matrix **with non-negative values. These are called the singular values of** A**.**

Claim 1.1.8 Let $A = U\Sigma V^\top$ be an SVD of a matrix A. It holds that the columns of U and the rows of $V^\top$ are the left- and right singular vectors of A, corresponding to the singular values present on the diagonal of Σ.

Suppose that $rank(A) = r$. This means that the number of non-zero singular values is r, and notice that $r \leq \min\{d, m\}$. When $m \leq d$ then A and Σ are both wide matrices (they have more columns than rows):

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

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

SVD of A, which we can write as:

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

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

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

Lemma 1.1.9 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 $A^\top A$.

This means that the eigenvalues of $AA^\top$ and $A^\top A$ equal to the square root of the singular values of A. In addition, as the orthogonal matrices of the EVD contain the eigenvectors of the matrix, the eigenvectors of $AA^\top$ are the left singular values of A while the eigenvectors of $A^\top A$ are the right singular values of A.

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

1.2 Multivariate Calculus

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

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

1.2.1 Derivatives, Gradients and Jacobians

1.2.1.1 Derivatives

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

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

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

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

neural networks. The derivative if this function is:

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

■

Note that at $x = 0$ the derivative of ReLU is undefined. We will discuss ways to handle such cases in chapter 8. Next, let us expand the derivative definition to multivariate functions.

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

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

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

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

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

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

■

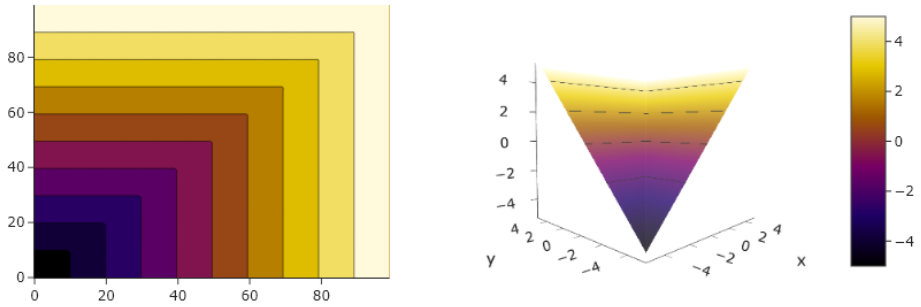


Figure 1.1: *Visualization of bi-variate max function in 3D and 2D. Chapter 1 Examples - Source Code*

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

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

■

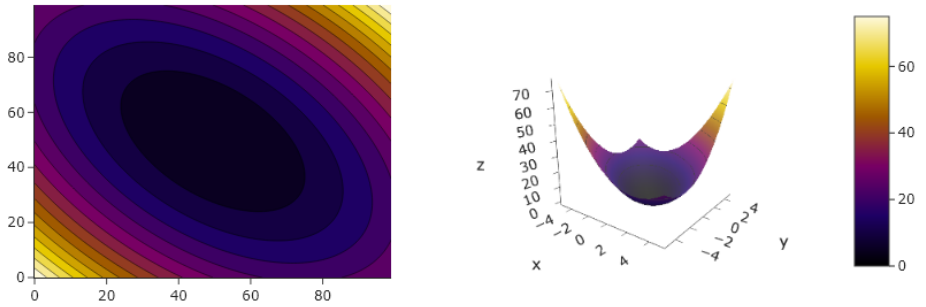


Figure 1.2: *Visualization of bi-variate polynomial of degree 2 in 3D and 2D. Chapter 1 Examples - Source Code*

1.2.1.2 Gradients

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

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

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

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

■

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

Proof. From linearity of the derivative:

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

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

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

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

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

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

1.2.1.3 Jacobians

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

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

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

$$J_{\mathbf{x}}(\mathbf{f}) \coloneqq \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_m} \\ \vdots & & \vdots \\ \frac{\partial f_d(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_d(\mathbf{x})}{\partial x_m} \end{bmatrix}$$

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

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

■

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

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

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

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

■

1.2.1.4 Chain Rules

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

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

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

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

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

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

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

■

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

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

This function is known as the Mean Square Error (MSE) and in subsection 2.1.2 we will use the above derivation in order to find the values of $\mathbf{x}$ that minimize f as above.

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

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

As the coefficients a_j are in the power of the exponent function, all outputted values are strictly positive. Moreover, since the numerator appears in the denominator summed up with some other positive numbers, $S_j < 1$. Therefore, it is in the range $(0,1)$. For example, the 3-element vector $(1,2,5)$ gets transformed into $(0.02, 0.05, 0.93)$. The order of elements by relative size is preserved,

and they add up to 1.0. In addition, the third element is now farther away from the first two, Its softmax value is dominating the overall slice of size 1.0 in the output.

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

Let us calculate the derivative of the softmax function. Denote $g_{i(\mathbf{a})} \coloneqq e^{a_i}$ and $h(\mathbf{a}) \coloneqq \sum_{k=1}^{N} g_i(\mathbf{a})$. 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 First Order Function Approximation

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Recall that $f(\mathbf{p}_0+\mathbf{p}) \approx f(\mathbf{p}_0) + \nabla f(\mathbf{p}_0) \cdot \mathbf{p}$. In addition, the angle between $\nabla f(\mathbf{p}_0)$ and $\mathbf{p}$ is $\nabla f(\mathbf{p}_0) \cdot \mathbf{p} = ||\nabla f(\mathbf{p}_0)|| \cdot ||\mathbf{p}|| \cos\theta$. As we are only interested in the direction in which to step, and not the step size, let us assume that $||(\mathbf{p})|| = 1$. Since $\cos x \in [-1,1]$, the direction that maximizes f is $\mathbf{p}_{max} := \nabla f(\mathbf{p}_0) / ||\nabla(\mathbf{p}_0)||$. Similarly the direction that minimizes f is $\mathbf{p}_{min} := -\nabla f(\mathbf{p}_0) / ||\nabla(\mathbf{p}_0)||$. So, if we adapt a step-wise approach, for sufficiently small enough steps sizes, for maximizing/minimizing f, moving in the direction of the gradient or opposite to the gradient is a good approach. We will revisit this concept in chapter 8

1.3 Probability and Statistics

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

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

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

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

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

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

Another question is how many participants should we contact? Intuitively, if we ask a very small set of participants our estimation of $\widehat{\mu}$ (and therefore our prediction of T would be very wrong. As we increase the number of participants asked, the *sample size*, we will get a more accurate estimation of $\widehat{\mu}$. To answer how accurate of an estimation, let us suppose that we know the true value of μ. Then we should look at the difference between our estimation and the true value, and how it changes as m changes. We can then choose an accuracy parameter $\varepsilon > 0$ and require that $|\widehat{\mu} - \mu| \leq \varepsilon$. By solving for m we will get a bound on the number of participants we should ask.

1.3.1 Fundamental Definitions

1.3.1.1 Probability Space

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

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

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

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

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

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

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

■ **Example 1.8** Suppose we throw two fair dice, then $\Omega = \{1,\ldots,6\}^2$ and $\mathcal{D}((i,j)) = \frac{1}{36}$ ■

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

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

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

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

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

■

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

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

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

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

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

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

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

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

■

1.3.1.2 Random Variables

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

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

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

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

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

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

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

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

■ **Example 1.9** A fair coin is tossed twice. The probability space is given by $\Omega = \{T,H\}^2$. As this is a fair coin then $\forall \omega \in \Omega\ \mathcal{D}(\omega) = \frac{1}{4}$. Let X denote the number of heads obtained in each outcome (ω). The possible values of X are 0, 1, and 2. The probability of each value is

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

■

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

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

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

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

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

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

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

1.3.1.3 Mean and Variance

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

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

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

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

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

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

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

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

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

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

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

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

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

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

■ **Example 1.10** Let Z be a random variable such that $P(Z=1) = p, P(Z=0) = 1-p$. The variance of Z is: $Var[Z] = \mathbb{E}\left[Z^2\right] - (\mathbb{E}[Z])^2 = p - p^2 = p(1-p)$. ■

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

Proof.

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

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

■

1.3.2 A Little Statistics: Mean and Variance Estimation

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

unknown distribution, we attempted to *estimate* the distribution's mean. Let us formulate what we have done above and estimate both the expected value and the variance.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

■

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

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

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

1.3.3 Multivariate Probabilities

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

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

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

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

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

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

1.3.3.1 Normal Distribution

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

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

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

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

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

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

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

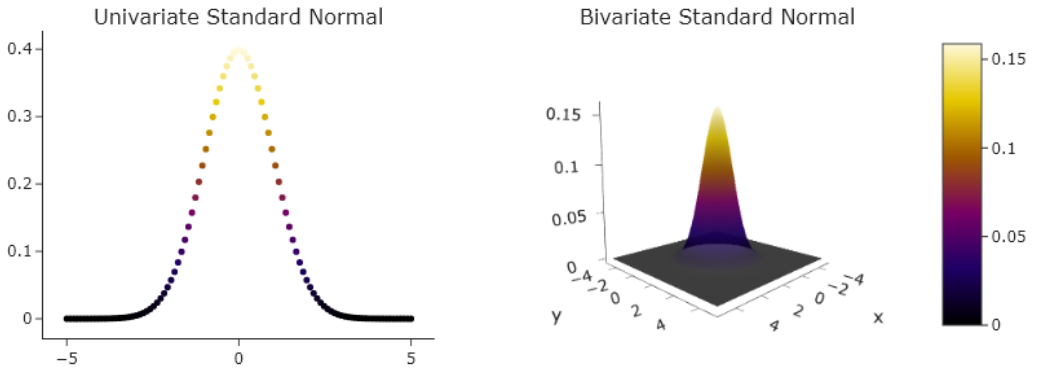


Figure 1.3: *Uni- and bivariate normal distributions.* *Chapter 1 Examples - Source Code*

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:

Definition 1.3.17 The ***Marginal distribution*** of a subset if a collection of random variables with

a joint probability distribution, is the probability distribution of the variables in the set:

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

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

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

Observe that we can write the PDF as follows:

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

Using the definition of the marginal distribution:

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In matrix notation the sample covariance matrix is given by:

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

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

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

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

Now, following the definition of the sample covariance matrix:

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

■

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 variances $\sigma_{X_1}^2 = \sigma_{X_2}^2 \equiv \sigma^2$ (Figure 1.4). 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 1.4 we can indeed see that there is no apparent relation between a points x coordinate and y coordinate. When we calculate the sample covariance matrix, as the sample size m increases, our estimation tends to $\widehat{\Sigma} = \sigma^2 I_2$.

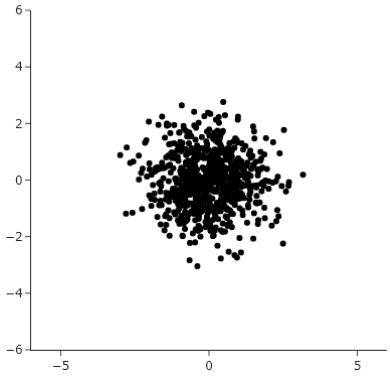


Figure 1.4: *Uncorrelated variables with identical variance.* *Chapter 1 Examples - Source Code*

Next, will us look at how linear transformations affect our data and the sample covariance matrix $\widehat{\Sigma}$ (Figure 1.5). We start transforming our data by multiplication by a scaling matrix:

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

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

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

.

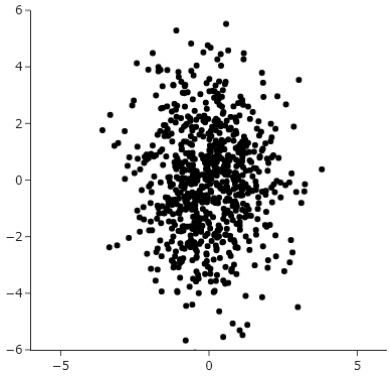


Figure 1.5: *Uncorrelated scaled variables with different scaling in the x and y axis. Chapter 1 Examples - Source Code*

Lastly, over the scaled data let us follow with a rotation transformation (Figure 1.6). 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 1.6 the transformed data shows a correlation between the x and y axes. To understand the reasoning behind let us calculate the sample covariance matrix of the scaled and then rotated data.

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

As the off diagonal element of the covariance matrix is non-zero, the two variables are correlated. Note that if we would not have applied an *asymmetric* scaling (i.e if $s_1 = s_2$), the off diagonal

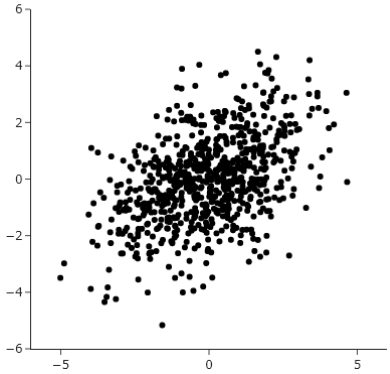


Figure 1.6: *Correlated variables produced by rotation of uncorrelated variables. Chapter 1 Examples - Source Code*

elements would have been zero. This means that rotation alone is not sufficient to induce correlations between random variables.

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

1.3.4 Probability Inequalities

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

■ **Example 1.11** Suppose we have a bag containing red and blue balls and we would like to estimate the fraction p of red balls in the bag by randomly picking one ball at a time and then putting it back in. The straightforward strategy is to draw m samples and then to use the following estimation ("prediction")

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

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

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

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

1.3.4.1 Markov's and Chebyshev's inequalities

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

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

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

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

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

■

Corollary 1.3.7 Let $X_1,\ldots,X_m$ be m i.i.d. non-negative random variables and denote the expectation value of each by $\mathbb{E}[X_i] = \mathbb{E}[X]$ $i = 1,\ldots,m$. Denote also $\overline{X} = \frac{1}{m}\sum_{i=1}^m X_i$. Then for any $a > 0$:

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

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

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

and therefore, we obtain the desired bound. ■

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

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

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

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

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

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

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

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

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

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

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

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

1.3.4.2 Coin Prediction Example

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

Formally, a coin flip is a Bernoulli random variable Z which takes the value 1 for "Heads" or 0 for "tails". We shall denote the probability distribution of Z by $\mathcal{D}_p$, where $0 \leq p \leq 1$ and $\mathcal{D}_p(1) = p, \mathcal{D}_p(0) = 1 - p$ are the probabilities to obtain 1 and 0 correspondingly. Let this be a fair

coin where $p = 1/2$. Let the following be a sequence of m tosses of this coin $Z_1, \ldots, Z_m \overset{iid}{\sim} Ber(p)$ and denote the results of the tosses (that is, the samples) by $S = (z_1, \ldots, z_m)$. Denote the probability of obtaining a specific S by $\mathcal{D}_p^m(S)$ (or simply by $\mathcal{D}^m(S)$).

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

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

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

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

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

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

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

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

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

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

The first condition means that regardless to the true value of p, it is enough to draw $m_{\mathcal{A}}$ samples (i.e., to toss the coin $m_{\mathcal{A}}$ times) in order to know p, with a certainty of $1 - \delta$ and an accuracy of $\pm\varepsilon$. The second conditions means that, at least for some values of p, drawing $m_{\mathcal{A}}(\varepsilon, \delta) - 1$ samples would

not be enough for that.

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

Choosing an Algorithm

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

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

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

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

Estimating Sample Complexity Using Markov's Inequality

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

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

If we select m to be

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

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

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

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:

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

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

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

Estimating Sample Complexity Using Hoeffding's Inequality

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

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

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

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

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

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

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

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

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

2. Linear Regression

Regression is a statistical method to model the relation between a set of explanatory variable (i.e features) and a scalar response, also referred to as dependent variable. Therefore, our *sample domain* is $\mathcal{X} = \mathbb{R}^d$ and our *response set* is $\mathcal{Y} = \mathbb{R}$. We assume that there exists some deterministic (unknown to us) function $f : \mathcal{X} \to \mathcal{Y}$ that is *underlying* the relation between each sample x and it's response y.

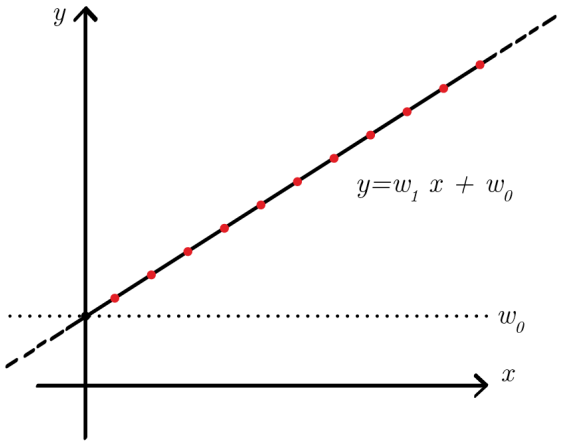


Figure 2.1: Linear Regression Illustration: *Illustration of a regression problem with a single explanatory variable. Samples represented with red dots.*

For reasons we discuss later (add reference to no free lunch), whenever we try to model such a relation, we restrict ourselves to specific families of function. These families are referred to as *hypothesis classes*. In the case of linear regression we aim to model the relation using a linear

function f. Formally, we define hypothesis class of linear regressors as the set of linear functions from the domain set to the response set:

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

Each function h in the class is characterized by the **weights** (or regression coefficients) $w_1,\ldots,w_d$ representing the d features and an **intercept** w_0. To simplify notation, for a given sample $\mathbf{x} = (x_1,\ldots,x_d)^\top \in \mathbb{R}^d$ we add a zero-th coordinate with the value of one, and define $\mathbf{x} = (1,x_1,\ldots,x_d)^\top \in \mathbb{R}^{d+1}$. Using this notation each function in the hypothesis class can be written in the form $h(\mathbf{x}) = \langle \mathbf{x}, w \rangle = \mathbf{x}^\top w$. From this point onward we will assume that the intercept is already incorporated into the weights vector and samples.

Thus, given a set of samples $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$also referred to as a *training set*, we are looking for some vector $\mathbf{w} \in \mathbb{R}^d$ such that $\forall i \in [m] \quad y_i = \mathbf{x}_i^\top w$. Let us arrange the data in a matrix form. We define a response *column vector* $\mathbf{y} \in \mathbb{R}^m$ and a samples *matrix* $\mathbf{X} \in \mathbb{R}^{m \times d}$ as follows (where rows represent samples and columns represent features):

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

The samples matrix $\mathbf{X}$ is referred to as the *design matrix*. Now we can write it as a system of m independent linear equations, which we want to solve for $\mathbf{w}$ (a set of d variables):

$$\mathbf{y} = \mathbf{X}\mathbf{w} \tag{2.2}$$

If there exists (at least one) solution, it means that f that created our data is indeed in $\mathcal{H}_{reg}$. This is called **The Realizable Case** or **The Realizability Assumption**. Later in this chapter we will discuss why the assumption that the responses $\mathbf{y}$ are *exactly* linear in the samples $\mathbf{X}$ is not realistic. Many times, even when the relation is indeed linear, the given data may be almost linear. In such cases f is *not* in $\mathcal{H}_{reg}$. This is called **The Non-Realizable Case** in which we must settle for finding $\widehat{f} \in \mathcal{H}_{reg}$ which is "good enough" for our purposes.

In the realizable case, the system of equations has either a unique solution or an infinite number of solutions, depending on the kernel of $\mathbf{X}$ being trivial or non-trivial (i.e $dim(Ker(\mathbf{X}))$ equals or not-equals to zero). As there exists a solution, we know that $\mathbf{y} \in Im(\mathbf{X})$. In the case of a trival kernel we conclude that the columns of X, meaning the features of our data, are linearly independent. In the non-realizable case, the system has no solution and therefore $\mathbf{y} \notin Im(\mathbf{X})$.

2.1 Ordinary Least Squares

2.1.1 Least Squares Loss Function

To design a learning algorithm that will learn (predict) f, we would like to be able to quantify the *risk* of choosing any predictor $\widehat{f} \in \mathcal{H}_{reg}$. We derive different algorithms, each with different properties,

by the choice of the risk function. In the case of Ordinary Least Squares, OLS, we choose to work with the mean squared risk (**??**):

$$\left.\begin{array}{l}\theta := \mathbf{y} \\ \widehat{\theta} := \widehat{\mathbf{y}}\end{array}\right\} \Rightarrow R(\mathbf{y},\widehat{\mathbf{y}}) = \mathbb{E}_{\widehat{\mathbf{y}}}\left[(\mathbf{y}-\widehat{\mathbf{y}})^2\right]$$

where $\widehat{\mathbf{y}}$ is the linear prediction of $\mathbf{y}$ when selecting some predictor from our hypothesis class:

$$\widehat{\mathbf{y}} = \widehat{f}(\mathbf{x}) = \mathbf{x}^\top \widehat{\mathbf{w}} \quad s.t \quad \widehat{\mathbf{w}} \in \mathcal{H}_{reg}$$

As we assess our chosen hypotheses using the squared risk it makes sense to choose $\widehat{f}$ that minimizes the same loss *on the training data we already have*. The function chosen to calculate the risk over a given dataset is called a **Loss Function** and is denoted by $L_S\left(\widehat{f}\right)$ where S is the set of samples $\widehat{f}$ is evaluated over.
Putting it all together, given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$and some prediction rule $\widehat{f} \in \mathcal{H}_{reg}$ the quantity of the squared risk over the training data is called the **empirical risk**. In our case the empirical risk of the linear function is given by:

$$R\left(y, \widehat{f}(\mathbf{x})\right) = \sum_{i=1}^{m}\left(y_i - \mathbf{x}_i^\top \widehat{\mathbf{w}}\right)^2 = ||\mathbf{y} - \mathbf{X}\widehat{\mathbf{w}}||_2^2 = (\mathbf{y} - \mathbf{X}\widehat{\mathbf{w}})^\top (\mathbf{y} - \mathbf{X}\widehat{\mathbf{w}})$$

2.1.2 Residual Sum of Squares

Therefore, under the selection of the squared risk function, minimizing the empirical risk 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 Euclidean distance. The deviation $y_i - \mathbf{x}_i^\top \mathbf{w}$ is called the i-th **residual**. The total empirical risk is therefore called **Residual Sum of Squares** (or **RSS**), denoted by:

$$RSS(\mathbf{w}) := ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 \tag{2.3}$$

As such, learning the lienar function by ERM is solving the quadratic problem:

$$\mathbf{w}^* := \underset{\mathbf{w}\in\mathbb{R}^d}{argmin}\, RSS(\mathbf{w}) = \underset{\mathbf{w}\in\mathbb{R}^d}{argmin}\, ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 \tag{2.4}$$

It is therefore a smooth function of $\mathbf{w}$ with a minimum (or minima) If X is of full rank then $||\mathbf{y} - \mathbf{X}\mathbf{w}||^2$ has a unique minimum. This minimum is the predictor $\widehat{f}$ we would like to find.

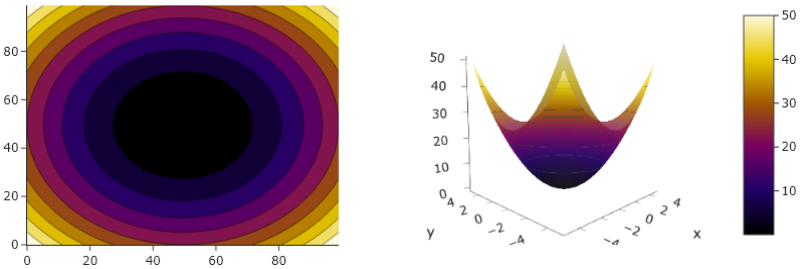


Figure 2.2: *Illustration of the RSS function in R^2 for X of full rank. Chapter 2 Examples - Source Code*

For $\mathbf{w}$ to be a minimizer of the RSS function all its partial derivatives should be zero. Recalling the definition of the inner product, this condition can be written as:

$$\begin{gathered}\frac{\partial}{\partial w_j} RSS(\mathbf{w}) = -2\sum_{i=1}^{m} (\mathbf{x}_i)_j \cdot \left(y_i - \mathbf{x}_i^\top \mathbf{w}\right) = 0 \quad j = 0, \ldots, d \\ \Downarrow \\ \nabla RSS(\mathbf{w}) = -2\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) = 0\end{gathered}$$

2.1.2.1 The Normal Equations

So a necessary condition for $\mathbf{w}$ to be a minimizer of the objective is to be a solution for the following linear system, known as the **Normal Equations**:

$$\mathbf{X}^\top (y - \mathbf{X}\mathbf{w}) = 0 \quad \iff \quad \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top X \mathbf{w} \tag{2.5}$$

Let us show that $\widehat{\mathbf{w}} = \left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top \mathbf{y}$ is the desired minimizer. Before proving so we will first make an assumption over our data $\mathbf{X}, \mathbf{y}$. We will assume that the different features (i.e columns of $\mathbf{X}$) are linearly independent. We will make this assumption in all of the models we will discuss in the course.

Theorem 2.1.1 — The Non-Singular Case. Let $\mathbf{X}$ be a design matrix with m observations and d explanatory variables, and $\mathbf{y}$ a response vector. If $d \leq m$ then $\widehat{\mathbf{w}} = \left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top \mathbf{y}$ is a unique minimizer of (2.4). That is, for any $\mathbf{w} \neq \widehat{\mathbf{w}} \quad ||\mathbf{y} - \mathbf{X}\mathbf{w}|| > ||\mathbf{y} - \mathbf{X}\widehat{\mathbf{w}}||$

Proof. As we would like to find a minimizer of the objective, we begin with equating the objective's derivative to zero:

$$\frac{\partial RSS(\mathbf{w}, \mathbf{X}, \mathbf{y})}{\partial \mathbf{w}_k} = \frac{\partial \left(\sum_{i=1}^{m} \left(\mathbf{y}_i - \mathbf{x}_i^\top \mathbf{w}\right)^2\right)}{\partial \mathbf{w}_k} = -2\sum_{i=1}^{m} \left(\mathbf{y}_i - \sum_{j=1}^{d} \mathbf{x}_{ij}\mathbf{w}_j\right) \mathbf{x}_k = 0$$

Equivalently, in matrix notation we get that:

$$\forall k \in [d] \quad \left[\mathbf{X}^\top \mathbf{y}\right]_k - \left[\mathbf{X}^\top \mathbf{X}\mathbf{w}\right]_k = 0 \quad \Longrightarrow \quad \mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top \mathbf{X}\mathbf{w}$$

As $d \leq m$ the design matrix $\mathbf{X}$ is of full column rank $rank(\mathbf{X}) = d$ and therefore has a trivial kenel. As such, $\mathbf{X}^\top \mathbf{X}$ too is of full rank. This means that the matrix $\mathbf{X}^\top \mathbf{X}$ is non-singular (i.e invertible) and $\left[\mathbf{X}^\top \mathbf{X}\right]^{-1}$ exists:

$$\begin{gathered}\mathbf{X}^\top \mathbf{y} = \mathbf{X}^\top \mathbf{X}\mathbf{w} \\ \Downarrow \\ \left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top \mathbf{y} = \left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top \mathbf{X}\mathbf{w} \\ \Downarrow \\ \widehat{\mathbf{w}} = \left[\mathbf{X}^\top \mathbf{X}\right]^{-1} \mathbf{X}^\top \mathbf{y}\end{gathered}$$

Lastly, we would like to show that $\mathbf{w}$, which we have found to be an extramum of the objective, is a minimum. Taking the second derivative with respect to the parameters:

$$\frac{\partial^2 RSS(\mathbf{w}, \mathbf{X}, \mathbf{y})}{\partial \mathbf{w}_k \partial \mathbf{w}_l} = \frac{\partial -2\sum_{i=1}^{m} \left(\mathbf{y}_i - \sum_{j=1}^{d} \mathbf{x}_{ij}\mathbf{w}_j\right) \mathbf{x}_k}{\partial \mathbf{w}_l} = 2\sum_{i=1}^{m} \mathbf{x}_k \mathbf{x}_l = 2\left[\mathbf{X}^\top \mathbf{X}\right]_{kl} \quad \forall k, l \in [d]$$

Notice that the matrix $\mathbf{X}^\top\mathbf{X}$ is a positive semi-definite matrix. Now, as we assumed that the columns of $\mathbf{X}$ are independent then for any $v \neq 0$ it holds that $v^\top \left[\mathbf{X}^\top\mathbf{X}\right] v = (\mathbf{X}v)^\top \mathbf{X}v = ||\mathbf{X}v||^2 > 0$ and $\mathbf{X}^\top\mathbf{X}$ is a positive definite matrix. Thus, $\widehat{\mathbf{w}}$ is indeed a minima of the RSS as requested. ■

In the case where $d > m$ the null space of $\mathbf{X}^\top\mathbf{X}$ is not trivial. Therefore the matrix is not invertible and there is an inifinite number of solutions.

Definition 2.1.1 Let $\mathbf{X} \in \mathbb{R}^{m\times d}$ and let $U\Sigma V^\top$ be its SVD. The **Moore-Penrose pseudoinverse** of $\mathbf{X}$ is $\mathbf{X}^\dagger = V\Sigma^\dagger U^\top$ where $\Sigma^\dagger$ is a $d \times m$ diagonal matrix defined by:

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

Theorem 2.1.2 — The Singular Case. Let $\mathbf{X}$ be a design matrix with m observations and d explanatory variables, and $\mathbf{y}$ a response vector. If $d > m$ then $\widehat{\mathbf{w}} = \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$ is a unique minimizer of (2.4)

Proof. As there are an infinite number of solutions let us begin with finding at least one. Let $\mathbf{X} = U\Sigma V^\top$ be the SVD of matrix $\mathbf{X}$. Therefore, U, V are orthogonal matrices and Σ is a $d \times m$ diagonal matrix where the diagonal elements $\Sigma_{i,i} \equiv \sigma_i, \quad \sigma_1 \geq \ldots \geq \sigma_d \geq 0$ are the singular values of $\mathbf{X}$.
Let $\Sigma^\dagger$ be a $d \times m$ diagonal matrix such that:

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

Notice that $\Sigma^\dagger\Sigma$ is too a diagonal matrix and if $\sigma_d > 0$ then $\Sigma^\dagger\Sigma = I_d$. Using $\mathbf{X}$'s SVD:

$$\begin{aligned} \mathbf{X}^\top\mathbf{X}\mathbf{w} &= \mathbf{X}^\top\mathbf{y} \\ \left(U\Sigma V^\top\right)^\top \left(U\Sigma V^\top\right)\mathbf{w} &= \left(U\Sigma V^\top\right)^\top \mathbf{y} \\ V\Sigma^\top U^\top U\Sigma V^\top\mathbf{w} &= V\Sigma^\top U^\top\mathbf{y} \\ \Sigma V^\top\mathbf{w} &= U^\top\mathbf{y} \\ \Sigma^\dagger\Sigma V^\top\mathbf{w} &= \Sigma^\dagger U^\top\mathbf{y} \end{aligned}$$

Therefore the solution is:

$$\widehat{\mathbf{w}} = \left(U\left(\Sigma^\dagger\right)^\top V^\top\right)^\top \mathbf{y} = U\Sigma^\dagger V^\top\mathbf{y} = X^{\dagger\top}\mathbf{y}$$

■

Corollary 2.1.3 $\widehat{\mathbf{w}} = \left[\mathbf{X}^\top\mathbf{X}\right]^{-1}\mathbf{X}^\top\mathbf{y}$ is always a solution of the Normal Equations

The estimator we found $\widehat{\mathbf{w}}$ is also referred to as the OLS estimator and is often noted as $\widehat{\mathbf{w}}^{OLS}$.

Neumerical Stability

Computers don't calculate over $\mathbb{R}$, they use bits and more specifically they use floating-point arithmetics with very finite precision. A lot of non-trivial knowledge and care are required in order to understand how learning algorithms are actually implemented in software. *Any* learning algorithm comes down to a lot of calculus (e.g gradients) and linear algebra (e.g. inverses) implemented in software. You should care *deeply* about how algorithms are implemented and when they break numerically, as in the following example.

Sometimes $\mathbf{X}^\top\mathbf{X}$ is formally invertible but *close to singular*. This happens if columns of $\mathbf{X}^\top$ are almost co-linear or if one column of $\mathbf{X}^\top$ is almost spanned by other columns. In this case some singular values of $\mathbf{X}$ will be nonzero, but very small. When this happens, Gauss elimination may yield wildly incorrect results. Also, because of double-precision arithmetics, $1/\sigma_i$ will not be precise. The practical solution in such cases is to choose a machine precision threshold, ε and let:

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

■ **Example 2.1** Let us find the OLS estimator over some mock scenario. Suppose we are interested in regressing running times in a $100m$ 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*. Let us fit an OLS model for this 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 OLS estimator is given by the closed form of $\widehat{\mathbf{w}} = \left(\mathbf{X}^\top\mathbf{X}\right)^{-1}\mathbf{X}^\top\mathbf{y}$. Over given data we obtain that $\widehat{\mathbf{w}} \approx (11.38, 0.003, -0.009)^\top$ (up to rounding up numbers).

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

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

■

2.1.3 A Statistical Model - Existence of Noise

So far we assumed that there exists some deterministic function $f : \mathcal{X} \to \mathcal{Y}$ that is underlying the relation between the domain- and response- sets. We would like to describe a more general probabilistic model, which better fits reality. Let us assume, as before, that the relation is linear in $\mathcal{X}$. In addition we add an additional element ε capturing random noise in the relation. This noise element is referred to as *error* and is a random variable. We make the assumptions that all errors are: (1) Centered (2) Uncorrelated (3) Have equal variance and (4) with a Gaussian distribution:

$$\forall i \in [m] \quad y_i = \mathbf{x}_i^\top \mathbf{w} + \varepsilon_i, \qquad \varepsilon_1, \ldots, \varepsilon_m \overset{iid}{\sim} \mathcal{N}\left(0, \sigma^2\right) \tag{2.6}$$

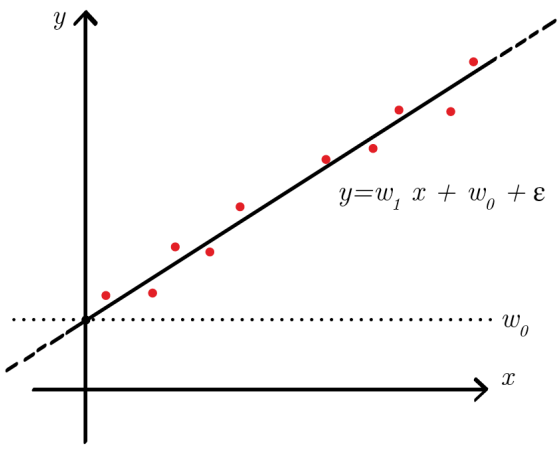


Figure 2.3: Probabilistic Model: *Modeling sample noise in the responses. Each sample is now found at* $(\mathbf{x}_i, y_i)$ *for* $y_i \sim \mathcal{N}\left(\mathbf{x}_i^\top \mathbf{w}, \sigma^2\right)$

As our model allows some uncertainty in the responses, even for identical $\mathbf{x}$s, we may end up with different ys. Let us adapt our learning algorithm from the deterministic case, and deal with the probabilistic (noisy) one. In the same manner as before, using the linear hypothesis class $\mathcal{H}_{reg}$, we asusme there exists an unknown coefficients vector $\mathbf{w}$. Denoting the noise vector $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_m)^\top$ we have in matrix notation:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon, \qquad \varepsilon \sim \mathcal{N}\left(0_d, \sigma^2 I_d\right) \tag{2.7}$$

Geometric Interpretation

To understand what is the meaning of the sample noise and what will the linear regression model do to the data, let us adapt a geometric interpretation. For $\mathbf{X} \in \mathbb{R}^{m \times d}$ the regression design matrix consider the space spanned by the columns of $\mathbf{X}$. Denote this subspace by $\mathcal{S} \coloneqq Col\left(\mathbf{X}\right) = Im\left(\mathbf{X}\right)$ and consider the OLS solution:

$$\widehat{w} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y}$$

For the predicted response vector $\widehat{\mathbf{y}}$ one of the following occurs:

- $\widehat{\mathbf{y}} \in \mathcal{S}$: This means that there exists a vector $\mathbf{w} \in \mathbb{R}^d$ such that $\mathbf{X}\mathbf{w} = \mathbf{y}$.
- $\widehat{\mathbf{y}} \in \mathcal{S}^\perp$: In this case $\mathbf{x}^\top \mathbf{y} = 0$.

- $\widehat{\mathbf{y}}$ is a combination of both. Namely we can write $\widehat{\mathbf{y}}$ as some combination $\widehat{\mathbf{y}} = \mathbf{y}_{\mathcal{S}} + \mathbf{y}_{\perp}$ where $\mathbf{y}_{\mathcal{S}} \in \mathcal{S}$ and $\mathbf{y}_{\perp} \in \mathcal{S}^{\perp}$.

For the third case we can then express $\widehat{\mathbf{y}}$ as:

$$\begin{aligned} \widehat{\mathbf{y}} &= \mathbf{y}_{\mathcal{S}} + \mathbf{y}_{\perp} \\ &= \mathbf{X}\mathbf{w} + 0 \\ &= \mathbf{X}\left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}\mathbf{y} \end{aligned} \tag{2.8}$$

Where the matrix $\mathbf{X}\left(\mathbf{X}^{\top}\mathbf{X}\right)^{-1}\mathbf{X}^{\top}$ is in-fact the **orgothonal projection matrix** onto the subpsace $\mathcal{S}$. Therefore, the additional sample noise is the component in the response that "pulls" y_i out of the subspace. When we perform the linear regression we project the samples onto the estimated subspace and remove the component estimated to be in the perpendicular subspace.

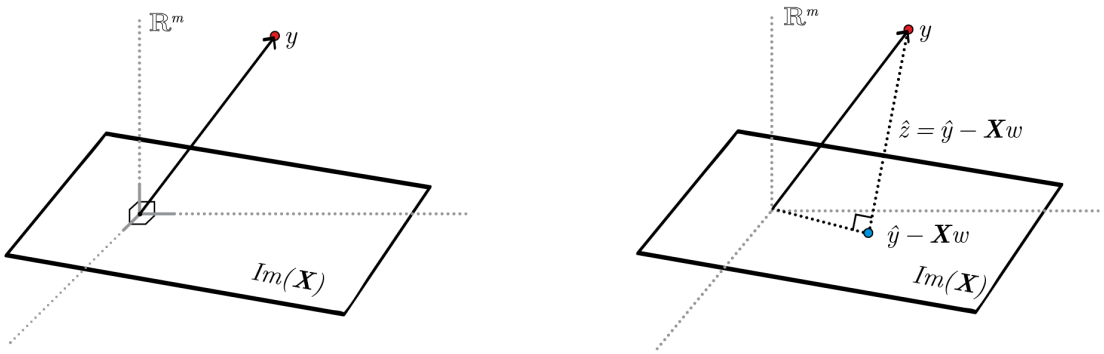


Figure 2.4: Linear Regression As Orthogonal Projection: *add caption*

2.1.4 Maximum Likelihood Equivalence

As seen above, different assumptions on the noise in our data yields different traits we can use for our advantage. Let us assume Gaussian noise $\varepsilon \overset{iid}{\sim} \mathcal{N}\left(0, \sigma^2\right)$. This means that the i-th observation is independently distributed $y_i \sim \mathcal{N}\left(\mathbf{x}_i^{\top}\mathbf{w}, \sigma^2\right)$. Suppose we **knew** the weight vector $\mathbf{w}$, we could then ask the following question: Given a fixed design matrix X and known coefficients vector $\mathbf{w}$, how likely is it to observe the response vector $\mathbf{y}$?

As we assumed Gaussian noise, we could answer this question using the density function of the Gaussian distribution. Therefore, the **likelihood** of $\mathbf{w}$, given the data is:

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

So solving for $\mathbf{w}$:

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

we conclude that the MLE (assuming Gaussian noise) is just the OLS estimator we obtained using the ERM principle over the squared loss.

2.1.5 Model Interpretation

Suppose we fitted a model $\mathbf{w}$ to a regression problem $\mathbf{X}, \mathbf{y}$ with the noise assumptions seen in 2.7. So:

$$\begin{aligned}&\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon \qquad\qquad \varepsilon \sim \mathcal{N}\left(0, \sigma^2 I_d\right) \\ &\widehat{\mathbf{w}} = \left[\mathbf{X}^\top \mathbf{X}\right]^{-1}\mathbf{X}^\top \mathbf{y}\end{aligned}$$

Let us think
Matan Begin with simple OLS. change in x -> expected (!) change in y Interpretation in multivariate OLS What can we learn from sign and scale of coefficients - and in general that scale of data matters

2.1.6 Categorical Variables

Consider the following problem. Suppose we want to predict the price of a house based on the following set of explanatory variables:

- The 'House Size' is a numeric variable accepting positive numbers.
- The 'Garden Size' is a categorical variable accepting the values: *small*, *medium* and *large*.
- The 'Number of Bedroons' is a categorical numeric variable accepting natural numbers.
- The 'House Type' is a categorical variable accepting the values: *private house*, *apartment* and $studio - apartment$.

We would like to construct the following regression problem:

$$y = \mathbf{x}^\top \mathbf{w}, \quad \mathbf{w} = \begin{bmatrix} w_{\text{house size}} \\ w_{\text{garden size}} \\ w_{\text{number of bedrooms}} \\ w_{\text{house type}} \end{bmatrix}$$

Notice that we have different *types* of variables, and it is not clear how to treat each one of them. The 'House Size' and 'Number of Bedrooms' variables are *quantitative* variables where we have a natural order over the values. That is, we are able to state if one house is larger than another or if the number of bedrooms in a house is less than the number of bedrooms in another house. For these variables, solving the set of linear equations seen in a linear regression problem is something we know how to do.

In the case of the 'Garden Size' variable, though the values are in fact not numeric (*small*, *medium* and *large*), we are able to define a logical order over these values. It makes sense to state that

$small < medium < large$ or that $large \neq small$. We can think of this order as some encoding of the non-numeric categories as numbers, with the order defined over these numbers being the order of the variable. For example: $small \mapsto 1, medium \mapsto 2, large \mapsto 3$.

Last, consider the 'House Type' variable. Can we define some logical map as we did for the 'Number of Bedrooms' variable? Are we able to state that in some sense that $studio-apartment > private-house$ or that $studio-apartment < private-house$? As we cannot find any logical ordering we must devise a different manner in which to deal with these variables. The most commonly used method is to encode these categories in what is known as *dummy variables* or *one-hot encoding*. Given a categorical variable with K categories we instead represent it as a binary vectors with K entries, where only one of these entries is "on".

$$\mathbf{x}_{\text{house type}} = \text{`}apartment\text{`} \quad \Rightarrow \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} \begin{matrix} \text{house size} \\ \text{garden size} \\ \text{number of bedrooms} \\ \text{private-house} \\ \text{apartment} \\ \text{studio-apartment} \end{matrix}$$

where $x_4 + x_5 + x + 6 = 1,\ x_4, x_5, \mathbf{x}_6 \in \{0,1\}$.

R Notice that by converting a single categorical variable with K categories to a one-hot encoding we are in-fact adding $K-1$ variables to our model. This addition has both influences on running times (as the algorithms we use are often polynomial in the number of features) and on the number of samples needed for learning (as we will see in add reference to sample complexity).

R add about one-hot creating features that are not independent from one another. and reference to regularization that this is tricky

2.2 Polynomial Fitting

Let us expand the model of linear regression. Suppose that instead of modeling the linear relation between $\mathcal{X}$ and $\mathcal{Y}$ as $y = \mathbf{x}^\top \mathbf{w} + \varepsilon$ we introduce a set of functions $h_1, \ldots, h_k$ such that $h_j : \mathbb{R}^d \to \mathbb{R}^k$. We then define the relation as:

$$y_i = \sum_{j=1}^{k} h_j(\mathbf{x})^\top \mathbf{w}_i$$

These functions $h_1, \ldots, h_k$, that are referred to as *basis functions*, enable us to describe a relation that is **linear in the parameters w** but could be non-linear in the original input $\mathbf{x}$. One specific case is of polynomial fitting. Let $x_1, \ldots, x_m \in \mathbb{R}$ and $y_1, \ldots, y_m \in \mathbb{R}$. We would like to describe a polynomial relation between $\mathcal{X}$ and $\mathcal{Y}$, of degree at most $k \in \mathbb{N}$. The hypothesis class is:

$$\mathcal{H}^k_{poly} = \left\{ x \mapsto p_{\mathbf{w}}(x) \,|\, \mathbf{w} \in \mathbb{R}^{k+1} \right\} \tag{2.9}$$

where $p_{\mathbf{w}}(x) = \sum_{i=1}^{K} w_i x^i$. In this case, we define the set of basis functions to be $h_j(x) = x^j$ for any $j \in \{0,\ldots,k\}$. Then denote $h(x) := (h_0(x),\ldots,h_k(x))^\top$. Thus, a polynomial of degree k is given by:

$$p_{\mathbf{w}}(x) = \sum_{j=0}^{k} h_j(x)\,\mathbf{w}_j = \langle h(x),\mathbf{w}\rangle = h(x)^\top \mathbf{w}$$

As before we want to find a coefficients vector $\mathbf{w}$. Given a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ we transform it to a sample $\widetilde{S} := \{(h(x_i), y_i)\}_{i=1}^m$, and solve the following LS problem:

$$\widehat{w} := \underset{\mathbf{w}\in\mathbb{R}^{k+1}}{argmin} \frac{1}{m}\sum_{i=1}^{m}\left(h(x_i)^\top \mathbf{w} - y_i\right)^2$$

Notice that the design matrix $\mathbf{X}$ defined over the transformed data is the Vandermonde matrix:

$$\mathbf{X} = \begin{bmatrix} — & h(x_1) & — \\ — & h(x_2) & — \\ & \vdots & \\ — & h(x_m) & — \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & \cdots & x_2^k \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^k \end{bmatrix}$$

Since we assume that the x_i's are different from one another, the design matrix $\mathbf{X}$ is of full rank. This means that solving this linear (in $\mathbf{w}$) system of equations can be done as we have seen for the non-singular case above.

R Here we have seen 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 3.3.4.

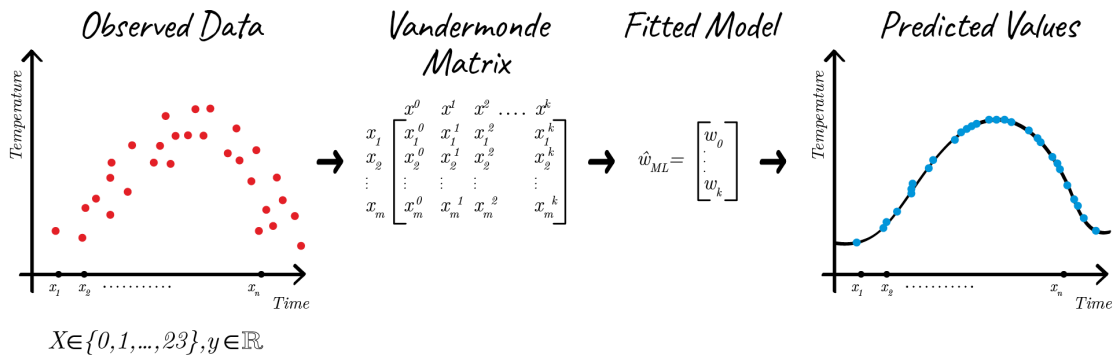


Figure 2.5: Scheme of Polynomial Fitting: *Dataset of hourly temperature where x_i denotes time of day and y_i the temperature. Fitting polynomial of degree* 4.

2.3 Bias and Variance of Estimators

Given a regression problem $\mathbf{X},\mathbf{y}$ we have seen how to solve $\mathbf{y}=\mathbf{X}\mathbf{w}+\varepsilon$. That is, finding a vector $\mathbf{w}$ that satisfies: $\mathbf{y}\approx\mathbf{X}\mathbf{w}$. We showed that the vector minimizing the sum of square distances is given by

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

As $\mathbf{y}$ is a random variable, the $\widehat{\mathbf{w}}^{ols}$ 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 2.3.1 Let $\widehat{\theta}$ be an estimator of θ. The *bias* of $\widehat{\theta}$ is the expected deviation between θ and the estimator: $B\left(\widehat{\theta}\right):=\mathbb{E}\left[\widehat{\theta}\right]-\theta$. $\widehat{\theta}$ is said to be *unbiased* if $B\left(\widehat{\theta}\right)=0$.

Definition 2.3.2 Let $\widehat{\theta}$ be an estimator of θ. The variance of $\widehat{\theta}$ is the expected value of the squared sampling deviations: $var\left(\widehat{\theta}\right):=\mathbb{E}\left[\left(\widehat{\theta}-\mathbb{E}\left[\widehat{\theta}\right]\right)^2\right]$.

What is the expectation in both the bias (2.3.1) and the variance (2.3.2) been calculated over? An estimator is a function over a sample $S=x_1,\ldots,x_m\in\mathbb{R}^d$ used to estimate some parameter: $\widehat{\theta}\left(x_1,\ldots,x_m\right)\overset{?}{\approx}\theta$. As such, the expectation of $\widehat{\theta}$ is over the selection of the samples. Going back to the $\widehat{\mathbf{w}}^{ols}$ 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 $\mathbf{y}=\mathbf{X}\mathbf{w}+\varepsilon$, $\varepsilon\sim\mathcal{N}\left(0,\sigma^2 I_d\right)$ and $\widehat{\mathbf{w}}$ being the OLS estimator. Show that $\widehat{\mathbf{w}}$ is an unbiased estimator. ■

Proof.

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

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

To get some intuition about these two properties, let us revisit polynomial fitting. Consider the polynomial $y=x^4-2x^3-0.5x^2+1+\varepsilon$ for $\varepsilon\sim\mathcal{N}\left(0,2\right)$. In Figure 2.6 we choose a set of x values and create 10 different datasets from this model, keeping the x values consistent between datasets but sampling the noise each time: $\left\{\left\{\left(\mathbf{x}_i,y\left(\mathbf{x}_i\right)+\varepsilon_{ij}\right)\right\}_{i=1}^m\right\}_{j=1}^{10}$, $\varepsilon_{ij}\overset{iid}{\sim}\mathcal{N}\left(0,2\right)$. Then, we fit a 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 $\mathbb{E}[\widehat{y}] \pm 2 \cdot Var(\widehat{y})$ for a given x, also known as the confidence interval. The wider this area is, the more out prediction of $\widehat{y}$ varies for different samples. This area captures the concept of the variance.

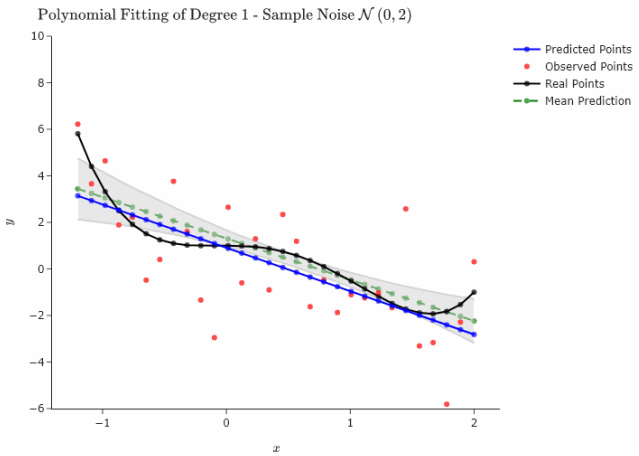


Figure 2.6: Polynomial Fitting: *Fitted polynomial of degree* 1 *over different datasets differing only in values of added sample noise. Chapter 2 Examples - Source Code*

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

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

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

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

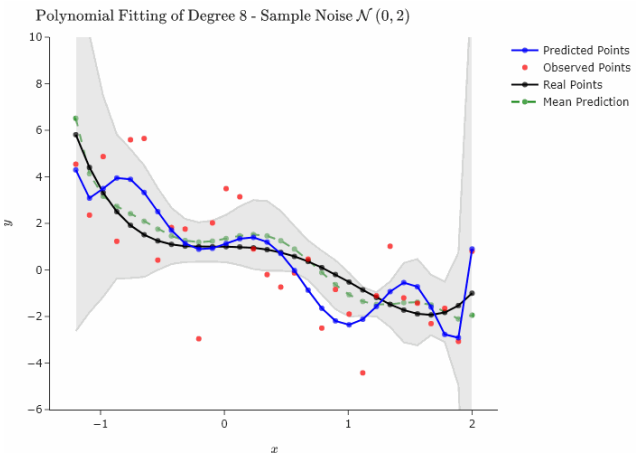


Figure 2.7: Polynomial Fitting: *Fitted polynomial of degree* 8 *over different datasets differing only in values of added sample noise.* *Chapter 2 Examples - Source Code*

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

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

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

3. Classification

3.1 Classification Overview

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

Throughout the chapter we discuss many different types of classifiers. For each type keep in mind the following guiding questions: (1) How does it model the classification problem? (2) What are the assumptions made on the data? (3) What is the learning principle we use? (4) How does the algorithm match the learning principle? (5) What is done in the training step and how to predict over new samples? (6) Is the model interpretable?

Classification Problems Examples

- Determine if a given network traffic pattern is one of a cyber attack or not.
- Detect fraud on credit card transactions.
- (Multi-class) What are the objects seen in a given picture.

■ **Example 3.1** Seen below are some samples of the "South Africa Heart Disease" dataset (See "Elements of Statistical Learning", section 4.4.2). Given the parameters of blood pressure, smoking,

family history, etc., could we predict who has/will have coronary heart disease (chd)? ■

	sbp	tobacco	ldl	adiposity	famhist	typea	obesity	alcohol	age	chd
1	160	12.00	5.73	23.11	Present	49	25.30	97.20	52	1
2	144	0.01	4.41	28.61	Absent	55	28.87	2.06	63	1
3	118	0.08	3.48	32.28	Present	52	29.14	3.81	46	0
4	170	7.50	6.41	38.03	Present	51	31.99	24.26	58	1
5	134	13.60	3.50	27.78	Present	60	25.99	57.34	49	1
6	132	6.20	6.47	36.21	Present	62	30.77	14.14	45	0
7	142	4.05	3.38	16.20	Absent	59	20.81	2.62	38	0
8	114	4.08	4.59	14.60	Present	62	23.11	6.72	58	1
9	114	0.00	3.83	19.40	Present	49	24.86	2.49	29	0
10	132	0.00	5.80	30.96	Present	69	30.11	0.00	53	1
11	206	6.00	2.95	32.27	Absent	72	26.81	56.06	60	1
12	134	14.10	4.44	22.39	Present	65	23.09	0.00	40	1
13	118	0.00	1.88	10.05	Absent	59	21.57	0.00	17	0
14	132	0.00	1.87	17.21	Absent	49	23.63	0.97	15	0
15	112	9.65	2.29	17.20	Present	54	23.53	0.68	53	0
16	117	1.53	2.44	28.95	Present	35	25.89	30.03	46	0
17	120	7.50	15.33	22.00	Absent	60	25.31	34.49	49	0

Figure 3.1: *South African Heart Data from ESL*

3.1.1 Loss Function

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

$$L_S(h) \coloneqq \sum_{i=1}^{m} \mathbb{1}\left[y_i \neq h(\mathbf{x}_i)\right] = \left|\{i | y_i \neq h(\mathbf{x}_i)\}\right| \tag{3.1}$$

3.1.2 Type-I and Type-II Errors

Though many times the mis-classification loss is indeed what we are looking for, when looking closer we recognize two kinds of errors.

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

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

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

■ **Example 3.3 — Drug safety.** Let us look at a more extreme example to help illustrate this point. We are creating a classifier to predicts whether a certain drug **is safe** to use for a particular person, or

unsafe/ deadly/ dangerous to use. We choose the labels such that -1 means "unsafe drug - do not use" and 1 means "safe drug - ok to use". Similar to before our errors are:

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

■

Therefore, we see that depending on the context of the classification problem, the two kinds of errors can have very different costs. We name the first error, the one we would like to avoid at all costs, the *Type-I error* and the second error as *Type-II error*. By choosing what label is "negative" and what label is "positive" we essentially defined what error is the Type-I error. As such, given a classification problem we try to choose the "negative" and "positive" labels such that the error we are more concern of (and therefore would like to avoid more) is the Type-I Error. That is, the error of misclassifying a negative sample by predicting it as a positive sample.

Table 3.1: *Classification Error Types*

Prediction \ Actual	**Negative**	**Positive**
Negative	TN	**FN** Type-II Error
Positive	**FP** Type-I Error	TP

3.1.3 Measurements of performance

Once we decided what are the "positive" and "negative" labels we can device an array of performance measurements for the binary classifier. Let us consider the following template, which will help us define the four basic terms seen in table 3.1:

The classifier {truthfully/falsely} stated {positive/negative}

So when the classifier **truthfully** states either "positive" or "negative" we understand that this statement (i.e prediction) was correct. In this case we are looking at the **True Positives (TP)** - positive samples that were classified as positives, and the **True Negatives (TN)** - negative samples that were classified as negatives. In the case the classifier **falsely** stated either "positive" or "negative" we understand that this statement (i.e prediction) was incorrect.

- If "the classifier *falsely* stated *positive*" we understand that the prediction was that the sample is positive, but in reality this is incorrect and therefore the true label is negative: $\widehat{y} = -1, y = 1$. These are the **False Positives (FP)**, which by convention we decided to define as the Type-I error.
- If "the classifier *falsely* stated *negative*" we understand that the prediction was that the sample is negative, but in reality this is incorrect and therefore the true label is positive: $\widehat{y} = 1, y = -1$. These are the **False Negatives (FN)**, which by convention we decided to define as the Type-II error.

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

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

3.1.4 Decision Boundaries

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

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

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

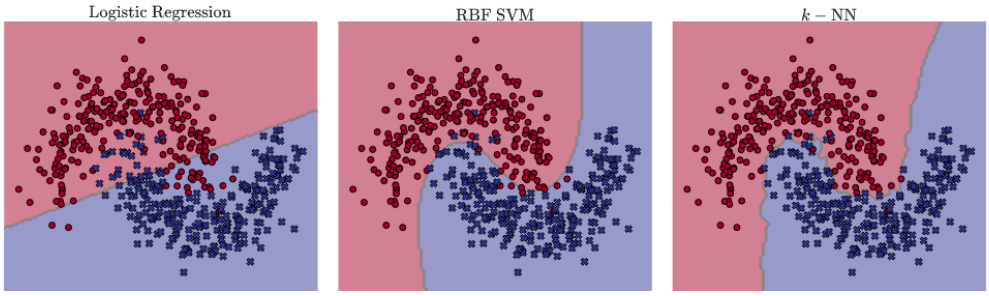


Figure 3.2: Decision Boundaries *of classifiers fitted over moons dataset. Chapter 3 Examples - Source Code*

3.1.5 ROC Curve

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

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

How do we choose α? There is an important **tradeoff** in the selection of α. If we set α to be very high we are mainly going to predict 0. By doing so we are **less** likely to have false-positives (which is the error we try to avoid at all cost), for which we are pleased. However, at the same time, we are bf more likely to gave false-negatives. So by setting α too high we might have low a FPR but "miss" (misclassify) most of the positive samples. On the other extreme, if we set α to be very low we are mainly going to predict 1. So we will be **more** likely to have false-positives, but at the same time will be **less** likely to have false-negatives. So if we set α too low, we might have a high FPR but "catch" (correctly classify) most of the positive samples. Therefore, we see that changing $\alpha \in [0,1]$ governs some trade-off between the chances of making a Type-I error (false-positives) and correctly classifying positive samples.

This trade-off was first studied during World War II, when radar was invented. The designer of the radar had to choose when to put a green dot on the radar, indicating a target detected there. Sometime radar waves would bounce off back from clouds or birds, and the designer had to choose a **threshold** α. If the radar pulse returning is stronger than α, the radar screen would show a green dot. If weaker than α, no dot. Now, if α is set too low (say $\alpha = 0.1$), the screen would be full of a thousand green dots - since any bird or cloud (with, say, $h(\mathbf{x} = 0.2)$) would be classified as **positive**, a target. So that the radar will be full of false positives, false targets, and will be useless. On the other hand, if α is set too high (say $\alpha = 0.9$) then enemy airplanes (with, say, $h(\mathbf{x} = 0.8)$) will not appear on the screen, since they will be classified by mistake as birds, and the radar again would be useless.

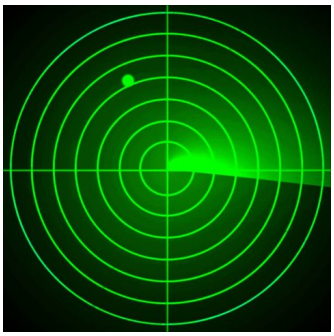


Figure 3.3: *Change to caricature of planes dressed up as birds/clounds - and a confused radar*

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”

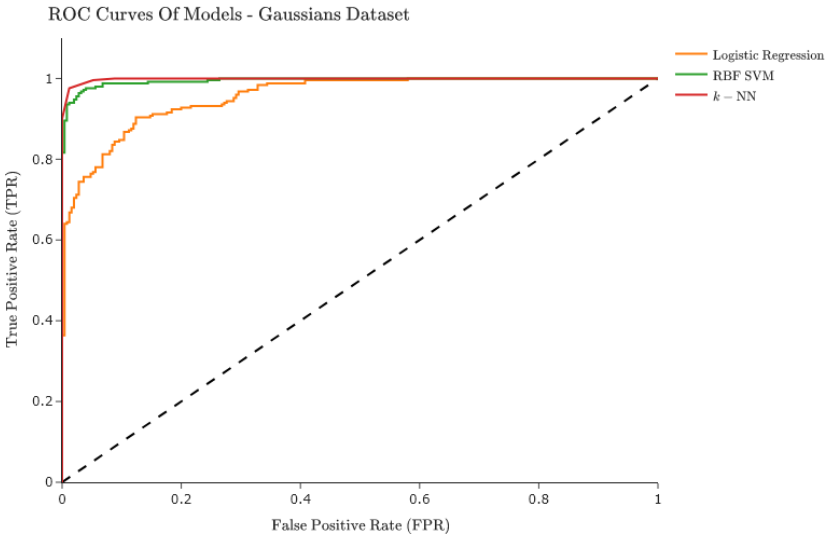


Figure 3.4: ROC Curve *of classifiers fitted over moons dataset. Chapter 3 Examples - Source Code*

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.
- **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.

3.2 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 3.5.

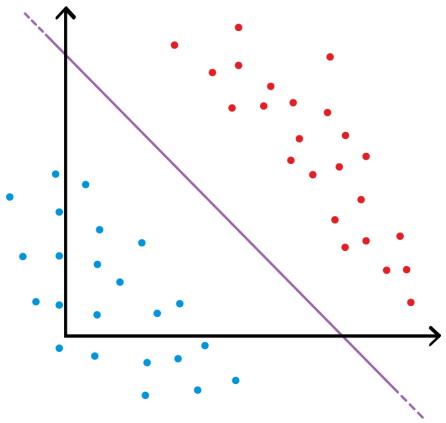


Figure 3.5: *Half-space Classification Illustration:*
For a domain-set $\mathcal{X} \in \mathbb{R}^2$ the two classes, coded as red and blue colors, are linearly separable

As we have seen in Equation 2.1, the family of linear functions can be described as:

$$L_d := \left\{ \mathbf{x} \mapsto \mathbf{x}^\top \mathbf{w} + b | \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \right\}$$

where the linearity refers to the functions being linear in the parameters $\mathbf{w}$. Next, consider the following definitions:

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

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

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

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

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

Notice that the family of linear functions L_d is in-fact the family of hyperplanes. As such, we can look at the family of functions that is the composition of the *sign* function and L_d, $sign \circ L_d$. This family defines the hypothesis class of the half-space classifiers. Denote $h_{\mathbf{w},b}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$ then:

$$\mathcal{H}_{half} := \left\{ h_{\mathbf{w},b}(\mathbf{x}) = sign\left(\langle \mathbf{x}, \mathbf{w} \rangle + b \right) | \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \right\} = \left\{ \mathbf{x} \mapsto sign\left(\langle \mathbf{x}, \mathbf{w} \rangle + b \right) \right\} \tag{3.2}$$

So why are functions in the form seen in 3.2 are half-space classifiers? Let us assume at first that

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

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

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

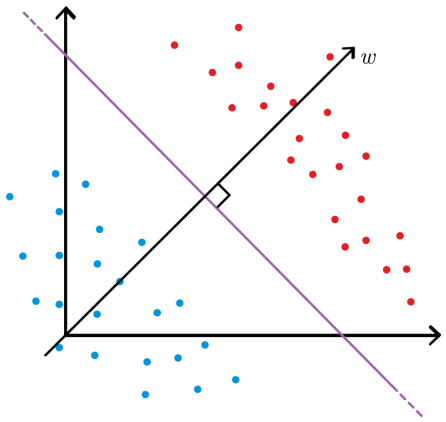


Figure 3.6: *Corresponding Hyperplane to* $\mathbf{w}^\perp$

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

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

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

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

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

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 $\mathcal{H}_{half}$.

3.2.1 Learning Linearly Separable Data Via ERM

To train a model over the defined hypothesis class of homogenous half-spaces ($\mathbf{w} \in \mathbb{R}^d, b = 0$) observe the following: For any hypothesis $h_{\mathbf{w}} \in \mathcal{H}_{half}$, the misclassified training samples are exactly

those where $y_i \cdot sign\left(\langle \mathbf{x}, \mathbf{w} \rangle\right) = -1$ or equivalently $y_i \langle \mathbf{x}, \mathbf{w} \rangle < 0$. So defining the loss of a given hypothesis over S is:

$$L_S\left(h_{\mathbf{w}}\right) := \sum_{i=1}^{m} \mathbb{1}\left[y_i \langle \mathbf{x}, \mathbf{w} \rangle < 0\right]$$

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

3.2.2 Computational Implementation

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

$$\underset{\mathbf{w} \in \mathbb{R}^d}{argmin}\, L_S\left(h_{\mathbf{w}}\right)$$

As we are assuming the given sample is linearly separable (realazibility), there exists a vector $\mathbf{w}_0 \in \mathbb{R}^d$ such that $y_i \cdot \langle \mathbf{x}, \mathbf{w}_0 \rangle > 0 \quad i = 1, \ldots, m$. This implies that there also exists a (different) vector $\mathbf{w}_1 \in \mathbb{R}^d$ such that $y_i \cdot \langle \mathbf{x}, \mathbf{w}_0 1 \rangle \geq 1 \quad i = 1, \ldots, m$ as we can simply normalize $\mathbf{w}_0$ by the smallest product:

$$\mathbf{w}_1 := \frac{1}{min_i \left\{y_i \cdot \langle \mathbf{x}, \mathbf{w}_0 \rangle\right\}} \cdot \mathbf{w}_0$$

Thus, it is enough to search for a vector $\mathbf{w}$ that satisfies $\forall i \in [m] \quad y_i \cdot sign\left(\langle \mathbf{x}, \mathbf{w} \rangle\right) \geq 1$. This is a problem of finding a vector that satisfies m linear constraints, and can be solved using generic linear programming solvers. We are interested in solving:

$$\begin{array}{ll} minimize & L_S\left(h_{\mathbf{w}}\right) \\ subject\,to & y_i \langle \mathbf{x}_i, \mathbf{w} \rangle \geq 1 \quad i = 1, \ldots, m \end{array} \tag{3.3}$$

which, as stated above for any ERM solution $L_S\left(h_{\mathbf{w}}\right) = 0$. Therefore, as this optimization problem has a trivial objective, it is a **feasibility** problem - we are looking for any vector which satisfies the constraints.

Proper writing+ deriving that this a a LP

3.2.3 The Perceptron Algorithm

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

Add convergence + correctness theorem

R The Perceptron algorithm is in fact a simple case of the more general algorithm of Subgradient Descent covered in chapter 8. Furthermore, we can modify the algorithm is such a way that rather than requiring an entire dataset S, it will each time get a single sample and update based on that one sample. We will encounter this variation in Online Learning chapter 9.

Algorithm 1 Batch-Perceptron

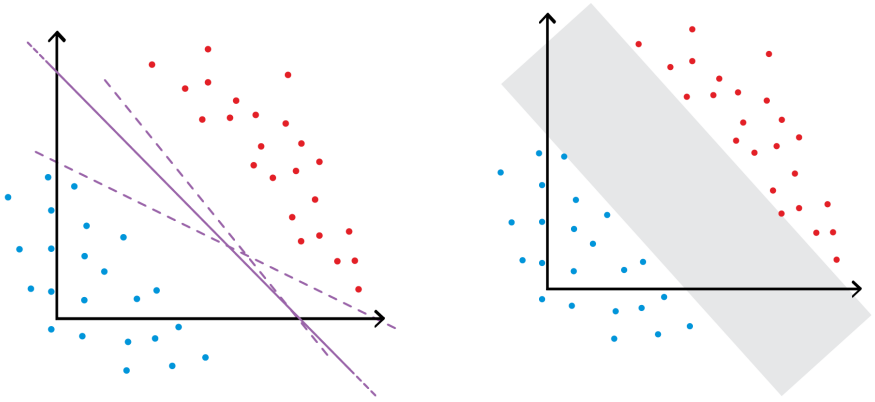
procedure PERCEPTRON($S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$)
  $\mathbf{w}^{(0)} \leftarrow 0$ ▷ Initialize parameters
  for $t = 1, 2, \ldots$ **do**
    if $\exists i \; s.t. \; y_i \left\langle \mathbf{w}^{(t)}, \mathbf{x}_i \right\rangle \leq 0$ **then**
      $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)}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

3.3 Support Vector Machines

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

- 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 3.3 and achieving the minimal empirical loss (of zero when assuming realizibility or any other positive number if not). As such we are faced with the problem of which one to choose. Figure 3.7 illustrates the existance of multiple separating hyper-planes.

- A more severe problem rises when we chose to work with the ERM learning principle for selecting the hypothesis, but the data is not linearly separable (non-realizible case). In this case the optimization problem described in 3.3 is computationally hard.



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

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

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

Returning to the same hypothesis class of non-homogeneous separating half-spaces $\mathcal{H}_{half}$ (3.2), let us **not** assume the data is linearly separable. We would like to describe a different learning principle that will be able to cope with both problems above: finds a unique hypoerplane and that

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

3.3.1 Maximum Margin Learning Principle

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

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

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

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

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

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

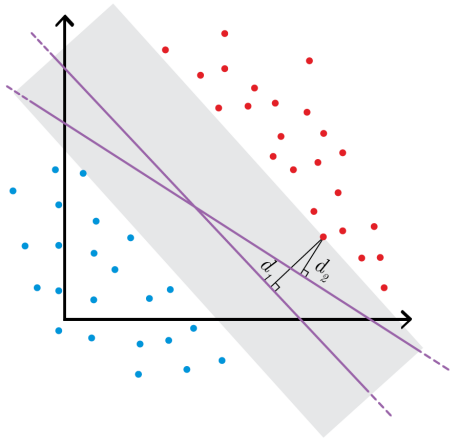


Figure 3.8: *Margin of specified hyper-planes*

3.3.2 Hard-SVM

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

$$\begin{array}{ll} \textit{maximize} & M((\mathbf{w}, b), S) \\ \textit{subject to} & y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 \quad i = 1, \ldots, m \end{array} \tag{3.4}$$

The optimization variables are $\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}$. Comparing with the linear program of half-spaces (3.3) we see that the constraints are kept, which ensure the hyperplane chosen separates the training sample, but instead of a trivial objective, we seek to minimize the margin (We don't worry about "maximize" instead of "minimize" as we can just multiply the objective by -1).

3.3.2.1 Convex Optimization

The Hard-SVM optimization problem 3.4 (and many other problems we will encounter throughout the course) is part of a wide family of **convex optimization problems**.

Definition 3.3.3 — Optimization Problem. An optimization problem over $\mathbb{R}^d$ has the general form:

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

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

Definition 3.3.4 — Convex Set. Let S be a vector space over the field of $\mathbb{R}$. A subset of S, $C \subseteq S$ is convex if and only if $\forall \mathbf{v}, \mathbf{u} \in C$ the line connecting $\mathbf{v}$ and $\mathbf{u}$ is included in C. Equivenantly it means that $\forall \alpha \in [0,1]$ it holds that the convex combination $(1-\alpha)\mathbf{v} + \alpha\mathbf{u}$ is included in C.

Definition 3.3.5 — Convex Function. A function $f : \mathbb{R}^d \to \mathbb{R}$ is a convex function of $dom(f)$ is a convext set in $\mathbb{R}^d$, namely:

$$\forall \alpha, \beta \in \mathbb{R} \; \forall \mathbf{v}, \mathbf{u} \in dom(f) \quad f(\alpha\mathbf{v} + \beta\mathbf{u}) \leq \alpha f(\mathbf{v}) + \beta f(\mathbf{u})$$

Then, naturally, a convex optimization problem is an optimization problem as above in which $f_0, f_1, \ldots, f_n$ are all convex functions. When these functions are all linear, this is a linear programming problem (as for example in 3.3).

Definition 3.3.6 — Quadratic Program. An optimization problem is called a Quadratic Program (QP) if it can be written in the following form:

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

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 there 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. One example for specialized solvers you've seen in

Algorithms course are specialized solvers for linear programs.

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 our learning algorithm chooses is given as the minimizer of some quantity (such as empirical risk). So implementation of the learning algorithm needs to solve an optimization problem.

Sometimes, our hypothesis class is equivalent to a Euclidean space (for example, in linear regression we saw that linear functions are determined by the weight vector and an intercept, so the hypothesis class of linear functions was equivalent to the Euclidean space $\mathbb{R}^{d+1}$, and we've just seen the same happens for the hypothesis class of half-space classifiers). 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. In chapter 8 we dive deeper into theory and algorithms of convex optimization.

3.3.2.2 Solving Hard-SVM

So is the Hard-SVM a convex optimization problem? Recall, that by our optimization problem 3.4, we are searching of a separating hyperplane that maximizes the margin from all points. As for any $c > 0$ it holds that $(\mathbf{w}, b) = (c\mathbf{w}, cb)$ we can w.l.o.g constraint ourselfs to $||\mathbf{w}|| = 1$. This way, each hyperlane has a unique vector $\mathbf{w}$ that corresponds to it.

Definition 3.3.7 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 $||w|| = 1$ 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. To solve this we begin with defining some point in the hyperplane, calculate it's distance from $\mathbf{x}$ and then showing minimality. Let $\mathbf{v} := \mathbf{x} - (\langle \mathbf{w}, \mathbf{x} \rangle + b) \cdot \mathbf{w}$. This point $\mathbf{v}$ is indeed in the hyperplane:

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

with a distance from x of:

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

Lastly, let us conclude that such $\mathbf{v}$ is the closest point in the hyperplane to $\mathbf{x}$. Let $\mathbf{u}$ be some point in

the hyperplane, then:

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

■

So, as the margin 3.3.2 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 infact of the form:

$$\begin{aligned}
&\underset{||\mathbf{w}||=1, b}{argmax} && \min_{i\in[m]} |\langle \mathbf{w}, \mathbf{x}_i\rangle + b| \\
&subject\,to && y_i \cdot (\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1 \quad i = 1, \ldots, m
\end{aligned} \tag{3.5}$$

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:

$$\begin{aligned}
&\underset{(\mathbf{w}, b)}{argmin} && ||\mathbf{w}||^2 \\
&subject\,to && y_i \cdot (\langle \mathbf{x}_i, \mathbf{w}\rangle + b) \geq 1 \quad i = 1, \ldots, m
\end{aligned} \tag{3.6}$$

Then, $\mathbf{w}^* := \gamma \mathbf{v}^*, b^* := \gamma c^*$ for $\gamma = ||\mathbf{v}^*||^{-1}$ is an optimal solution for 3.3.2.2.

Proof. Let us begin with simplifying Equation 3.3.2.2. Consider a *feasible* solution $\mathbf{w}$ to the problem (that is, statisfies all constraints). It holds that $|\langle \mathbf{w}, \mathbf{x}_i\rangle + b| = y_i(\langle \mathbf{w}, \mathbf{x}_i\rangle + b)$. Hence, we can rewrite Equation 3.3.2.2 as:

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

■

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

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

Next, let us examine Equation 3.6. Notice that an optimal solution $\mathbf{v}^*, c^*$ will always satisfy:

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

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 3.6 it must satisfy all constraints and is therefore a feasible solution for $(*)$ with an objective:

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

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

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

But then $(\mathbf{w}_2/\delta, b_2/\delta)$ is a feasible solution for 3.6 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 3.6 is a quadratic program for which there exist efficient solves. By using them to solve problem 3.6 we can obtain an optimal solution for the Hard-SVM optimization problem.

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

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

where support vectors satisfy $y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)$ 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. If that is not the case then the optimization problem has no solutions as for any candidate $(\mathbf{w}, b)$ at least one of the constraints $y_i \cdot \left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) \geq 1$ cannot be satisfied.

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

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

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

$$\begin{aligned} &minimize && ||\mathbf{w}||^2 \\ &subject\,to && \begin{cases} y_i \cdot \left(\langle \mathbf{x}_i, \mathbf{w}\rangle + b\right) \geq 1 - \xi_i & i = 1, \ldots, m \\ \xi_i \geq 0 \quad \wedge \quad \frac{1}{m}\sum_{i=1}^m \xi_i \leq C \end{cases} \end{aligned} \tag{3.7}$$

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

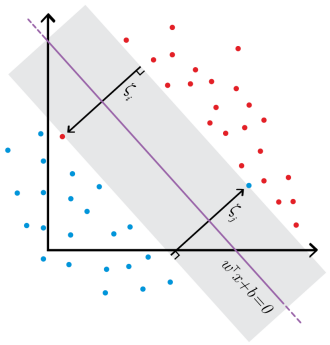


Figure 3.9: *Slack variables of data-points that are on the "wrong" side of the hyper-plane.*

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

$$\begin{array}{ll} \underset{\mathbf{w},\{\xi_i\}}{minimize} & \lambda \left|\left|\mathbf{w}\right|\right|^2 + \frac{1}{m}\sum_{i=1}^{m} \xi_i \\ subject\,to & y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i,\ \xi_i \geq 0 \quad i = 1, \ldots, m \end{array} \tag{3.8}$$

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

$$\ell^{hinge}(a) = \max\{0, 1-a\},\ a \in \mathbb{R} \tag{3.9}$$

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 (3.8) is equivalent to

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

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

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

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

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

The hyper-parameter λ controls the trade-off between the two 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. If we choose to work with this optimization problem to choose h, the constant λ also moves us along different members of a family of learners, each with a different bias-variance tradeoff. λ is known as a **regularization parameter**. This topic is covered in section 6.1.

3.3.4 Kernel Methods

The three classifiers seen above (halfspace classifier, Hard-SVM and Soft-SVM) all use the halfspaces hypothesis class. Though these linear classifiers are fairly simple to implement, the hypothesis class of halfspaces is of limited power. Consider the case where $\mathcal{X} = \mathbb{R}$ with the sample seen in Figure 3.10 (left). As the data in not linearly separable, all three learners will fail. Notice however, that if instead of using the original feature representation of the data, we embed the samples in a new feature space, such as mapping each sample $x \to \left(x, x^2\right)$, we get a linearly separable sample in $\mathbb{R}^2$ (right).

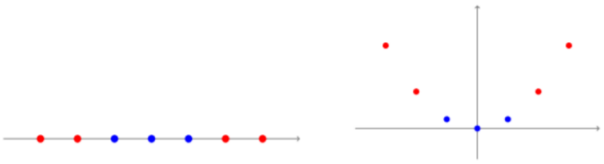


***Figure 3.10: Mapping to Feature Space**: Data originally linearly inseparable that is linearly separable in mapped feature space*

So, to enrich the expressiveness of an hypothesis class, we can consider embedding the data into another (potentially of higher dimension) feature space, over which we then learn some predictor. Schematically:

- Given some domain set $\mathcal{X} \subseteq \mathbb{R}^d$ and a learning algorithm $\mathcal{A}$, select an embedding $\psi : \mathcal{X} \to \mathcal{F}$ for some feature space $\mathcal{F}$. In many cases we will want $\mathcal{F} \subseteq \mathbb{R}^k$ for some $k \gg d$.
- Train $\mathcal{A}$ using the training set $\{(\psi(\mathbf{x}_i), y_i)\}_{i=1}^{m}$.

This approach seems very attractive as the assumption that our data is linearly separable in *some* feature space does not look very limiting. In order to apply this strategy, several challenges need to be addressed:

- **Selection of ψ**: We need to find an appropriate mapping ψ under which the data is linearly separable.
- **Computational complexity**: Often, as $k \gg d$ (and sometimes even infinite) and as ψ could be very complicated, computing $\psi(\mathbf{x})$, is computationally expensive.
- **Sample complexity**: When transitioning to a potentially much higher feature space, the number of samples needed to learn a halfspace increases. Add reference to VCdim of halfspaces

■ **Example 3.4** Suppose we are interested in separating between people that got a certain disease and people that don't, based on gene expression levels. We are given $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where $\mathbf{x}_i$ is a vector in $\mathbb{R}^{20000}$ (the "gene space") where each coordinate $\mathbf{x}_i(j)$ denotes how much is gene j expressed in individual i. $y_i \in \{\text{healthy}, \text{sick}\}$ is the label. It is often the case that different genes are co-expressed, and modeling this co-expression might increase our expressive power and our ability to separate between the two groups. Let us create new additional features $\mathbf{x}_i(j)\mathbf{x}_i(k)$ that captures *interaction* between the j,k features. Using this mapping each sample gets the form:

$$\psi(\mathbf{x}) = (\mathbf{x}(1), \ldots, \mathbf{x}(20000), \mathbf{x}(1)\mathbf{x}(1), \ldots, \mathbf{x}(1)\mathbf{x}(20000), \ldots, \mathbf{x}(20000)\mathbf{x}(2000))^\top$$

Meaning that the feature space is of dimension $d + d^2/2 = \mathcal{O}(d^2)$ where $d = 20000$ the dimension of the original sample space. If we decide to use co-expression of geen trios then the feature space is of dimension $d + d^2/2 + d^3/3 = \mathcal{O}(d^3)$. Depending on the problem it might make sense to also look at feature interactions of even higher orders, suppose interactions of k features. If that is the case, our mapping function, maps us to a space of size $\mathcal{O}(d^k)$, which is exponential in the original sample space size. ■

3.3.4.1 Kernel Functions

As the mapping enables us to transition to a new feature space of very high dimension (could be exponential or even of infinite dimension), we are facting a computational challange in computing this transition. To address this problem we could use kernel based learning. In this approach, we decide to use a mapping function out of a (wide) specific family of mapping functions, called *kernel functions*, that even though might map to a very high dimension, can still be computed efficiently.

Definition 3.3.8 — PSD Kernel. A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a Positive Semi-Definite kernel on $\mathcal{X}$ if $\forall \{x_1, \ldots, x_m\} \subseteq \mathcal{X}$ the associated kernel matrix (also known as Gram matrix) $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is PSD.

Corollary 3.3.3 Let $g : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $h : \mathbb{R}^d \to \mathbb{R}^k$ such that $g(x,y) = h(x)^\top h(y)$. Then g is a PSD kernel function.

Proof. Given a arbitrary set $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subseteq \mathcal{X}$ for arbitrary size $m \in \mathbb{N}$ denote H the matrix where:

$$H = \begin{bmatrix} | & & | \\ h(\mathbf{x}_1) & \cdots & h(\mathbf{x}_m) \\ | & & | \end{bmatrix}$$

Consider the matrix $G = H^\top H$ so $G_{ij} = h(\mathbf{x}_i)^\top h(\mathbf{x}_j) = g(\mathbf{x}_i, \mathbf{x}_j)$. So G is the Gram matrix of g over the given set. This matrix is PSD as we were able to show it as the multiplication of $H^\top H$ add reference to PSD equiv. conditions ■

This corollary means that any kernel function can be written as an **inner product** of some transformation h of the data. This transformation h is the mapping function described above. So to check if a given function is a kernel function we could: (1) show the associated Gram matrix is PSD or (2) find the transformation h that satisfies that $g(x,y) = h(x)^\top h(y)$

Exercise 3.2 Let $\mathcal{X} = \mathbb{R}^d$. Show that $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is defined by $k(x,y) = 1$, $x,y \in \mathcal{X}$ is a valid kernel function. In addition, state a possible mapping function h such that $g(x,y) = h(x)^\top (y)$. ■

Proof. Let $\{\mathbf{x}_i\}_{i=1}^m \subseteq \mathcal{X}$. The Gram matrix of k over this set is: $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = 1$. As the eigenvalues of K are $m, 0 \geq 0$ and K is a symmetric matrix, we conclude that $K \in PSD$. Thus k a valid kernel function.
As for the mapping function h, it must satisfy that $h(x)^\top h(y) = 1$ for any x,y. So we could choose $h : \mathbb{R}^d \to \mathbb{R}^k$ that always returns some unit vector v. Then $h(x)^\top h(y) = v^\top v = ||v|| = 1$. Notice that we haven't specificed k, the dimension of the embedded space, which could be of any size we desire. ■

One commonly used kernel function is the polynomial kernel.

Claim 3.3.4 — Polynomial Kernels. Let $\mathcal{X} = \mathbb{R}^d$ and consider the k degree polynomial mapping:

$$\mathbf{x} \mapsto \left(1, \ldots, \mathbf{x}(i), \ldots, \mathbf{x}(i)\mathbf{x}(j), \ldots, \prod_{\substack{i \in J \\ J \subset [d], |J| = k}} \mathbf{x}(i) \right)$$

This is a valid kernel and can be computed by

$$k(\mathbf{x}, \mathbf{x}') := \left(1 + \langle \mathbf{x}, \mathbf{x}' \rangle\right)^k$$

Proof. To show this is a valid kernel, we will find some mapping ψ such that $K(\mathbf{x}, \mathbf{x}')$ equals to $\psi(\mathbf{x})^\top \psi(\mathbf{x}')$. For simplicity let $x_0 = 1 = x_0'$. Then:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^k \\ &= \left(\sum_{i=1}^d x_i x_i'\right)^k \\ &= \sum_{J \in \{0,\ldots,d\}^k} \prod_{i=1}^k x_{J_i} x'_{J_i} \\ &= \sum_{J \in \{0,\ldots,d\}^k} \prod_{i=1}^k x_{J_i} \prod_{i=1}^k x'_{J_i} \\ &= \sum_{J \in \{0,\ldots,d\}^k} \psi(\mathbf{x})_J \, \psi(\mathbf{x}')_J \\ &= \psi(\mathbf{x})^\top \psi(\mathbf{x}') \end{aligned}$$

Where we define $\psi : \mathbb{R}^d \to \mathbb{R}^{(n+1)^k}$ such that each coordinate of $\psi(\mathbf{x})$ corresponds to some $J \in \{0,\ldots,d\}^k$ and is equal to $\prod_{i=1}^k x_{J_i}$. Therefore we obtained that $k(\mathbf{x}, \mathbf{x}') = \langle \psi(\mathbf{x}), \psi(\mathbf{x}') \rangle$ ■

■ **Example 3.5** Let $\mathbf{x}, \mathbf{x}' \in \mathcal{X} = \mathbb{R}^2$ with $x_0 = 1 = x_0'$ and $k = 2$ so:

$$\begin{aligned} (1 + \langle \mathbf{x}, \mathbf{x}' \rangle)^2 &= (1 + x_1 x_1' + x_2 x_2')^2 \\ &= 1 + 2x_1 x_1' + 2x_2 x_2' + (x_1 x_1')^2 + (x_2 x_2')^2 + 2x_1 x_1' x_2 x_2' \end{aligned}$$

By defining ψ as $\psi(\mathbf{x}) = \left(1, 1 \cdot x_1, x_1 \cdot 1, 1 \cdot x_2, x_2 \cdot 1, x_1^2, x_2^2, x_1 \cdot x_2, x_2 \cdot x_1\right)^\top$ we achieve that $K(\mathbf{x}, \mathbf{x}') = \psi(\mathbf{x})^\top \psi(\mathbf{x}')$ ■

Claim 3.3.5 — Gaussian Kernels. The following is a valid kernel function:

$$k(\mathbf{x},\mathbf{x}') := \exp\left(-\frac{1}{2\sigma^2}\left|\left|\mathbf{x}-\mathbf{x}'\right|\right|^2\right), \quad \sigma^2 \in \mathbb{R}_+$$

with the mapping function ψ defined such that:

$$\forall n \in \mathbb{N} \quad \psi(\mathbf{x})_n := \frac{1}{\sqrt{n!}}\exp\left(-x^2/2\sigma\right)x^n$$

Proof. Let us begin with showing that $k(\mathbf{x},\mathbf{x}') = \psi(\mathbf{x})^\top \psi(\mathbf{x}')$:

$$\begin{aligned}
\psi(\mathbf{x})^\top \psi(\mathbf{x}') &= \sum_{n=0}^{\infty}\left(\tfrac{1}{\sqrt{n!}}\exp\left(-x^2/2\sigma\right)x^n\right)\left(\tfrac{1}{\sqrt{n!}}\exp\left(-\left(\mathbf{x}'\right)^2/2\sigma\right)\left(\mathbf{x}'\right)^n\right) \\
&= \exp\left(-\tfrac{\mathbf{x}^2+(\mathbf{x}')^2}{2\sigma}\right)\sum_{n=0}^{\infty}\tfrac{(\mathbf{x}\mathbf{x}')^n}{n!} \\
&= \exp\left(-\tfrac{1}{2\sigma^2}\left|\left|\mathbf{x}-\mathbf{x}'\right|\right|^2\right)
\end{aligned}$$

■

Interestingly, the feature space mapped by ψ is of infinite dimension.

Since a function is a valid kernel function if it's associated Gram matrix is PSD we can derive several closure properties for kernel functions. Here are some of these properties.

Claim 3.3.6 Let k be some valid kernel function then the following are valid kernel functions:

1. $k(x,y)x^\top Ay$, where $A \in PSD$.
2. $c \cdot k_1(x,y)$ where $c > 0$.
3. $exp(k_1(x,y))$
4. $f(x)k_1 f(y)$

Exercise 3.3 Using the closure properties above, show that the Gaussian kernel is a valid kernel function. ■

Proof.

$$\begin{aligned}
k(\mathbf{x},\mathbf{x}') &= \exp\left(-\left|\left|\mathbf{x}-\mathbf{x}'\right|\right|^2/2\sigma^2\right) \\
&= \exp\left(-\left|\left|\mathbf{x}\right|\right|^2/2\sigma^2\right)\cdot\exp\left(-\mathbf{x}^\top\mathbf{x}'/\sigma^2\right)\cdot\exp\left(-\left|\left|\mathbf{x}'\right|\right|^2/2\sigma^2\right) \\
&\overset{(*)}{=} f(\mathbf{x})\exp\left(-\mathbf{x}^\top\mathbf{x}'/\sigma^2\right)f(\mathbf{x}')
\end{aligned}$$

where we define $f(\mathbf{x}) = \exp\left(-\left|\left|\mathbf{x}\right|\right|^2/2\sigma^2\right)$. Now notice that this is a valid kernel as: (1) $\mathbf{x}^\top\mathbf{x}'$ is a valid kernel. (2) scaling by $\frac{1}{\sigma^2}$ is a valid kenel. (3) exponent of a valid kernel is a valid kernel. And finaly, (4) multiplying by $f(\mathbf{x}), f(\mathbf{x}')$ from left and right is a valid kernel. ■

R The Gaussian kernel is a specific case of the wider Radial Basis Function (RBF) kernel $k(\mathbf{x},\mathbf{x}') := \exp\left(-\beta\left|\left|\mathbf{x}-\mathbf{x}'\right|\right|^2\right)$.

Notice in both examples, the polynomial- and Gaussian kernels, the embedding is of much higher dimension but computing the value can be done efficiently without actually transitioning to the feature space. In the case of the k degree polynomial kernel, the feature space is of dimension $\mathcal{O}\left((d+1)^k\right)$. In the case of the Gaussian kernel the feature space is of an infinite dimension. However in both cases, computing the kernel function can be done in $\mathcal{O}(d)$.

3.3.4.2 The Kernel Trick

Now that we have introduced the notion of kernel functions, and seen that they encapsulate the mapping to some possibly-elaborate feature space, the next question is when and how can we apply it to different learning algorithms. Here we will use kernels in the context of the SVM algorithm and in 7.1.2 we use kernels in the context of the Kernel-PCA algorithm.
Add SVM using kernels - for that we need to state representer theorem
Finish with: The algorithm must be written as inner products of the data

3.4 Logistic Regression

3.4.1 A Probabilistic Model For Noisy Labels

Recall the statistical model for linear regression when assuming Gaussian errors (2.1.3), $\mathbf{y} = \mathbf{X}\mathbf{w} + \boldsymbol{\varepsilon}$ where $\boldsymbol{\varepsilon} \sim \mathcal{N}\left(0, \sigma^2 I_d\right)$. Notice that as $\boldsymbol{\varepsilon}$ is a random variable, $\mathbf{y}$ too is a random variable distributing as a multi-variate Gaussian:

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

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

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

where we also condition on $\mathbf{w}$ 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 being 1:

$$p\left(y_i|\mathbf{x}_i\right) = Ber\left(y_i|p\left(\mathbf{x}_i\right)\right)$$

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

$$p\left(y_i|\mathbf{x}_i,\mathbf{w}\right) = Ber\left(y_i|\phi_{\mathbf{w}}\left(\mathbf{x}_i\right)\right), \quad \phi_{\mathbf{w}} := sigm\left(\mathbf{w}^\top \mathbf{x}\right) \tag{3.10}$$

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

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

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

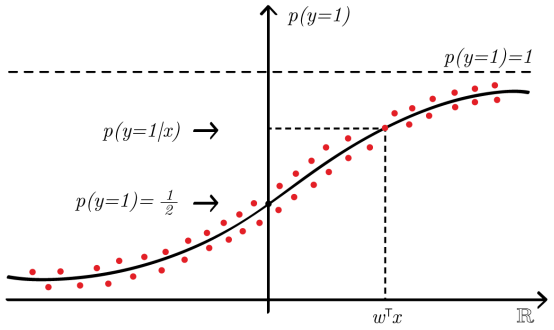


Figure 3.11: Illustration of fitted logit function *for values corresponding to* $\mathbf{w}^\top x$.

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

R In 3.10 we modeled the logistic regression model for binary classification problems. Notice that the Bernoulli distribution can be seen as a private case of the Multinomial distribution $Multinomial\left(p_1,\ldots,p_K\right), \sum_p i = 1, 0 \leq p_i \leq 1$. 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 $\sigma\left(\mathbf{a}\right) = e^{\mathbf{a}_i} / \sum_{j=1}^{K} e^{\mathbf{a}_j}$

3.4.1.1 The Hypothesis Class

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

$$\mathcal{H}_{logistic} \coloneqq \left\{ h_{\mathbf{w}}\left(\mathbf{x}\right) = sigm\left(\mathbf{w}^\top \mathbf{x}\right) | \mathbf{w} \in \mathbb{R}^{d+1} \right\} \tag{3.11}$$

However, notice that defining the hypotheses in such manner means that $h_{\mathbf{w}} : \mathbb{R}^{d+1} \to [0,1]$ and not $h_{\mathbf{w}} : \mathbb{R}^{d+1} \to \{0,1\}$ as required for classification problems. Since $\{0,1\} \subset [0,1]$, we can use the training sample to select a function in $\mathcal{H}_{logistic}$. This means we will be able to train a model, but how will we predict over new samples? Suppose our learner chose some $h_{\mathbf{w}} \in \mathcal{H}_{logistic}$. As we think about $h_{\mathbf{w}}\left(\mathbf{x}\right)$ as an estimate of the probability that the label corresponding to $\mathbf{x}$ is 1, we can use it for classification. If $h_{\mathbf{w}}\left(\mathbf{x}\right)$ is low the label is likely to be 0. If $h_{\mathbf{w}}\left(\mathbf{x}\right)$ is high, the label is likely to be 1. Choosing some **cutoff** value $\alpha \in [0,1]$, our class prediction will be: $\widehat{y} \coloneqq \mathbb{1}\left[h_{\mathbf{w}}\left(\mathbf{x}\right) > \alpha\right]$. To choose a fitting value for α we can calculate the Type-I and Type-II errors of the classifier and plot its ROC curve (subsection 3.1.2 and subsection 3.1.5)

3.4.1.2 Learning Via Maximum Likelihood

Once we have defined the logistic regression model (3.10) and hypothesis class (3.11), we would like to come up with a learner. To do so we will use the maximum likelihood principle. Recall, that by the maximum likelihood principle, we estimate the parameters (the desired hypothesis) as those that have the highest probability, given the data.

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

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

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:

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

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

$$\widehat{\mathbf{w}} := \underset{\mathbf{w}\in\mathbb{R}^{d+1}}{argmax} \sum_{i=1}^m \left[y_i \cdot \mathbf{w}^\top \mathbf{x}_i - \log\left(1+e^{\mathbf{w}^\top \mathbf{x}_i}\right)\right] \tag{3.12}$$

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 3.12 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 $\widehat{\mathbf{w}}$, as logistic regression is a very useful learner, there is a custom iterative algorithm that usually converges quickly to $\widehat{\mathbf{w}}$. This algorithm is based on **Newton-Raphson** iterations.

3.4.3 Interpretability

One important properly of the logistic regression learner is interpretability both in the sense of which features were important for the model and why was a certain prediction given. When working with $\mathcal{X} = \mathbb{R}^d$, we gather many feature, and might think that some of them are important for prediction while others less. Similar to linear regression, we are able to ask "which features were important" for the model by simply investigating the entries of the fitted coefficients vector $\mathbf{w}$. Feature 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 $\widehat{y}$. 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 $\mathbf{w}^\top \mathbf{x}$ will be larger, increasing the likelihood of the prediction being 1. If these values are of opposite signs to the coefficients then the expression $\mathbf{w}^\top \mathbf{x}$ will become smaller, decreasing the likelihood of the prediction being 1.

3.5 Bayes Classifiers

The logistic regression classifier seen above assumes a distribution on the labels $\mathcal{Y}$ given the samples Matan - need to be written - Begin with defining $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$ Then add notions of posterior(likelihod), prior and joint distributions

3.5.1 Bayes Optimal Classifier

From the above derive Bayes optimal

3.5.2 Naive Bayes

From the above derive Naitve optimal

3.5.3 Linear Discriminant Analysis

The Linear Discriminant Analysis (LDA) algorithm is yet another realization of the Bayes Optimal classifier. This time we assume that the points from each label have a different Gaussian distribution. Explicitly, we assume the following generative model:

1. Each sample selects a label $y_i = Ber(\pi)$, $\pi \in [0,1]$.
2. Then, the sample itself is drawn from the conditional probability of $X|Y$ where X denotes the random variable of sampling some samples $X = x$ and Y denotes the random variable of $Y = y$, $y \in \{0,1\}$. The distribution used to model $X|Y$ is a Gaussian distribution where each label is characterized by a different mean vector $\mu_0, \mu_1 \in \mathbb{R}^d$ but the same covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$.

Namely, for any $i \in 1, \ldots, m$ we assume that:

$$\begin{aligned} y_i &\sim Ber(\pi) \\ x_i|y_i &\sim \mathcal{N}(\mu_{y_i}, \Sigma) \end{aligned} \tag{3.13}$$

Under these assumptions, predicting the class of a new sample is done by simply using the Bayes Law over the Bayes Optimal classifier to get:

$$\widehat{y}(\mathbf{x}) := \underset{y}{argmax}\, \mathbb{P}(y|\mathbf{x}) = \underset{y}{argmax}\, \frac{\mathbb{P}(\mathbf{x}|y)\,\mathbb{P}(y)}{\mathbb{P}(]x)}$$

Claim 3.5.1 Let $\mathcal{D}$ be a joint distribution over $\mathcal{X} \times \mathcal{Y}$ with the conditional distribution

$$y|x \sim \mathcal{N}(\mu_k, \Sigma) \quad k \in \{0,1\}$$

Denote $\mathbb{P}(y=k) = \pi_k$ where $\pi_i \leq 0$, $\sum \pi_i = 1$. Then the Bayes Optimal classifier is given by:

$$\widehat{y}(\mathbf{x}) := \underset{k\in\{0,1\}}{argmax}\, a_k^\top \mathbf{x} + b_k, \quad a_k := \Sigma^{-1}\mu_k, b_k := -\frac{1}{2}\mu_k\Sigma^{-1}\mu_k$$

Proof. We begin with expressing $\mathbb{P}(y=k|\mathbf{x})$. By applying Bayes Law then:

$$\mathbb{P}(y=k|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y=k)\mathbb{P}(y=k)}{\mathbb{P}(\mathbf{x})} = \frac{\mathbb{P}(\mathbf{x}|y=k)\mathbb{P}(y=k)}{\sum_{k'\in\{0,1\}}\mathbb{P}(\mathbf{x}|y=k')\mathbb{P}(y=k)} = \frac{\pi_k\mathcal{N}(\mathbf{x}|\mu_k,\Sigma)}{\sum_{k'\in\{0,1\}}\pi_{k'}\mathcal{N}(\mathbf{x}|\mu_k',\Sigma)}$$

Next, from the PDF of a multivatiate Gaussian (1.3.16) then:

$$\begin{aligned}
\mathbb{P}(y=k|\mathbf{x}) &= \frac{\pi_k \cdot \frac{1}{Z}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)^\top\Sigma^{-1}(\mathbf{x}-\mu_k)\right)}{\sum_{k'}\pi_{k'}\cdot\frac{1}{Z}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_{k'})^\top\Sigma^{-1}(\mathbf{x}-\mu_{k'})\right)} \\
&= \frac{\pi_k \cancel{\frac{1}{Z}}\exp\left(\cancel{-\frac{1}{2}\mathbf{x}^\top\Sigma^{-1}\mathbf{x}} + \mathbf{x}^\top\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k\Sigma^{-1}\mu_k\right)}{\sum_{k'}\pi_{k'}\cancel{\frac{1}{Z}}\exp\left(\cancel{-\frac{1}{2}\mathbf{x}^\top\Sigma^{-1}\mathbf{x}} + \mathbf{x}^\top\Sigma^{-1}\mu_{k'} - \frac{1}{2}\mu_{k'}\Sigma^{-1}\mu_{k'}\right)} \\
&= \frac{\pi_k\exp\left(\mathbf{x}^\top\Sigma^{-1}\mu_k - \frac{1}{2}\mu_k\Sigma^{-1}\mu_k\right)}{\sum_{k'}\pi_{k'}\exp\left(\mathbf{x}^\top\Sigma^{-1}\mu_{k'} - \frac{1}{2}\mu_{k'}\Sigma^{-1}\mu_{k'}\right)}
\end{aligned}$$

for $Z := \sqrt{(2\pi)^d|\Sigma|}$ the Gaussians' normalization factor. Notice, that as we assume the classes are generated from Gaussians with the same covariance matrix $\frac{1}{Z}$ does not depend on k. Denote $a_k := \Sigma^{-1}\mu_k, b_k := -\frac{1}{2}\mu_k\Sigma^{-1}\mu_k$ and we conclude that:

$$\mathbb{P}(y=k|\mathbf{x}) \propto \exp\left(a_k^\top\mathbf{x} + b_k\right) \quad \Longrightarrow \quad \widehat{y}(\mathbf{x}) = \underset{k\in\{0,1\}}{argmax}\, a_k^\top\mathbf{x} + b_k$$

■

The claim above, besides showing that under the LDA assumptions we are dealing with a Bayes classifier, also tells us something about the classifier learned. Looking at the expression derived from the assumptions, we see that the classification is in fact done by some **linear separator/discriminant**.

It can be shown, by taking the $\log - likelihood$ ratio between the likelihood for being classified for a class divided the likelihood for being classified for the other class, that the decision boundary between the classes is linear, similar to Figure 3.12.

Learning A LDA Classifier

So, in order to predict using an LDA classifier we need to know the class probabilities π, the means μ_0, μ_1 and the covariance matrix Σ. To do so we derive the maximum likelihood estimators. Given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ then the likelihood is given by:

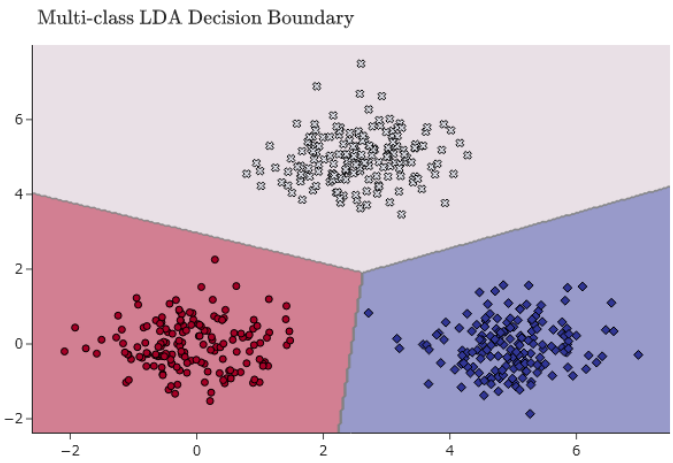


Figure 3.12: LDA Decision Boundaries *for a multiclass setup of three Gaussians.* *Chapter 3 Examples - Source Code*

$$\begin{aligned}\mathcal{L}(\Theta|\mathbf{X},\mathbf{y}) &= \mathbb{P}(\{(\mathbf{x}_i,y_i)\}_{i=1}^m|\Theta)\\ &\overset{iid}{=} \prod\mathbb{P}(\mathbf{x}_i,y_i|\Theta)\\ &= \prod\mathbb{P}(\mathbf{x}_i|y_i)\,\mathbb{P}(y_i|\Theta)\\ &= \prod\pi_{y_i}\mathcal{N}(\mathbf{x}_i|\mu_{y_i},\Sigma)\end{aligned}$$

where $\Theta := \{\pi,\mu_0,\mu_1,\Sigma\}$. By taking the derivative, each time with respect to π, μ_0, μ_1 and Σ we find that the MLE are:

$$\widehat{\pi}_y^{MLE} := \frac{1}{m}\sum\mathbb{1}[y_i=1] = \frac{m_y}{m} \tag{3.14}$$

$$\widehat{\mu}_y^{MLE} := \frac{1}{m_y}\sum\mathbb{1}[y_i=y]\,\mathbf{x}_i \tag{3.15}$$

$$\widehat{\Sigma}^{MLE} := \frac{1}{m}\sum_y\sum_{i:y_i=y}\left(\mathbf{x}_i-\widehat{\mu}_{y_i}^{MLE}\right)\left(\mathbf{x}_i-\widehat{\mu}_{y_i}^{MLE}\right)^\top \tag{3.16}$$

for m_y, $y\in\{0,1\}$ denoting the number of samples seen each of the classes. Looking closely at the expressions derived we in fact realize that the MLE predicts the values proportional to what is found in the training set.

R The covariance estimator described in 3.18 is the one derived when equating the derivative with respect to Σ to zero. This is a *biased* estimator. The unbiased estimator for the covariance matrix is given by multiplying by $\frac{1}{m-2}$ instead of $\frac{1}{m}$:

$$\widehat{\Sigma}^{MLE} := \frac{1}{m-2}\sum_y\sum_{i:y_i=y}\left(\mathbf{x}_i-\widehat{\mu}_y^{MLE}\right)\left(\mathbf{x}_i-\widehat{\mu}_y^{MLE}\right)^\top$$

This covariance matrix expression is known as the *pooled covariance* and it accounts for the use of an estimator of $\widehat{mu_j}$ rather the true parameter μ_j. In the general case of multi-classification the multiplication factor of the unbiased estimator is $\frac{1}{m-K}$ where K is the number of classes.

3.5.4 Quadratic Discriminant Analysis

In the LDA algorithm we assumed the data is generated from a set of Gaussians, differing in their mean but sharing the same covariance matrix (3.13). The Quadratic Discriminant Analysis algorithm allows different covariance matrices. That is, for any $i \in 1,\ldots,m$ we assume that:

$$\begin{aligned} y_i &\sim Ber(\pi) \\ x_i|y_i &\sim \mathcal{N}(\mu_{y_i}, \Sigma_{y_i}) \end{aligned} \tag{3.17}$$

By enabling different covariance matrices the quadratic expression (in $\mathbf{x}$) of $\mathbb{P}(y=k|\mathbf{x})$ does not cancel out. This in turn causes the decision boundaries between classes to be quadratic rather than linear. In both Figure 3.12 and Figure 3.13 the same data was used to fit either the LDA or the QDA models. We can see that while the decision boundaries of the LDA fit (Figure 3.12) are linear, in the case of QDA (Figure 3.13) we get curved (quadratic) boundaries.

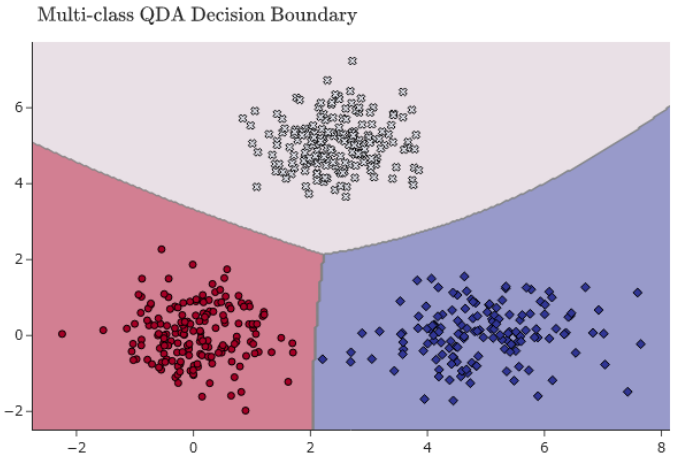


Figure 3.13: QDA Decision Boundaries *for a multiclass setup of three Gaussians. Chapter 3 Examples - Source Code*

Learning A QDA Classifier

Fitting a QDA classifier is very similar to the process of fitting a LDA classifier. The difference is in the estimation of the covariance matrices. In this case we fit a different covariance matrix for each class based on the samples of the class:

$$\widehat{\Sigma}_y^{MLE} := \frac{1}{m_y} \sum_{i:y_i=y} \left(\mathbf{x}_i - \widehat{\mu}_y^{MLE}\right)\left(\mathbf{x}_i - \widehat{\mu}_y^{MLE}\right)^\top \tag{3.18}$$

3.6 Nearest Neighbors

Nearest Neighbors learners predict the label of a new sample based on a set of "nearset" training samples. This means that it has no hypothesis class and no training stage per se. As prediction is based on some subset of training samples, that is determined by "closeness" to some newly given sample, the training step of the learner consists only of storing the training data so it is available for the learner when performing predictions.

This family of leaners are part of a wider **graph-based approach** for learning. In this approach we first define some graph structure over the samples - forming the nodes of the graph. Then, using the constructed graph we perform training and prediction. The different learners differ in how to define edges in the graph? are they weighted or not? how to transition between nodes? and how is this structure used for training and prediction?

3.6.1 Prediction Using k-NN

Let us begin with the simplest form of k-NN: given a training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ and some distance function $\rho : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$, prediction for a new sample $\mathbf{x}'$ is done by:

- Compute distance from $\mathbf{x}'$ with respect to ρ: $\forall \mathbf{x} \in S \quad d_{\mathbf{x}} := \rho(\mathbf{x}', \mathbf{x})$.
- Denote $\pi = (\pi_1, \ldots, \pi_m)$ the permutation of $(1, \ldots, m)$ such that $d_{\mathbf{x}_{\pi_1}} \leq \ldots \leq d_{\mathbf{x}_{\pi_m}}$
- Select k nearest samples $\mathbf{x}_{\pi_1}, \ldots, \mathbf{x}_{\pi_k}$ and predict by majority vote:

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

3.6.2 Computational Implementation

Implementing a k-nearest-neighbors classifier is very easy on small datasets, but becomes computationally challenging (either in terms of execution time or in terms of space) when d and/or m are large. There are generally three types of implementation approaches:

- **Brute force implementation:** We keep the entire training sample S in storage during the entire prediction process. For each new test sample $\mathbf{x} \in \mathbb{R}^d$ we calculate $\rho(\mathbf{x}, \mathbf{x}_i)$ $i \in [m]$ and partially sort to find the k smallest distances.
  Suppose ρ is the Euclidean distance. What are the computational costs of prediction? As the sample space is $\mathbb{R}^d$ computing the distance between two points is $\mathcal{O}(d)$. Doing so for all points in the dataset is $\mathcal{O}(dm)$. Next we want to retrieve the k nearest train samples. If $k \ll m$ we can retrieve k times the sample of minimal distance (without repeating previously selected samples) in a time complexity of $\mathcal{O}(km)$. However, if $k \approx$ .... then it is more computationally efficient to sort all distances and then select the k minimal. Lastly, summation over selected samples is done in $\mathcal{O}(k)$. All together the time complexity of such approach is $\mathcal{O}(dm + km)$. In terms of space complexity we must store distances of all training samples and thus $\mathcal{O}(m)$.

- **Exact nearest neighbors search with preprocessed data structure:** Depending on the selection of ρ, we could pre-process the training sample and construct a special data structure. After doing so in the training step, we can use this data structure to quickly locate the k nearest neighbors of a given test sample. In the case of ρ being the Euclidean distance we could you an algorithm such as *kd-tree*.

- **Fast randomized nearest neighbors search:** Beyond the scope of this course. add a single explanation sentence - or remove bullet all together

3.6.3 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.

Add 3 decision boundaries of 1,k,m-NN classifiers

As we change k we change the bias-variance tradeoff.

KNN - misclassification as function of k

3.7 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.

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.

3.7.1 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 noth, the hypothesis class consisted of half-spaces where prediction was determined by position of sample with respect to the hyper-plane. For decision trees we will describe a more complicated piecewise-constant prediction rule (more complex hypotheses). Let us consider a rule that partitions the sample space $\mathbb{R}^d$ into **axis-parallel boxes, or "hyper-rectangles"** where each box is associated with labels 1 or -1. The learner's task would be to use the training sample to "chop" the samples space $\mathcal{X}$ int a disjoint union of axis-parallel boxes, and to assign a class prediction to each box.

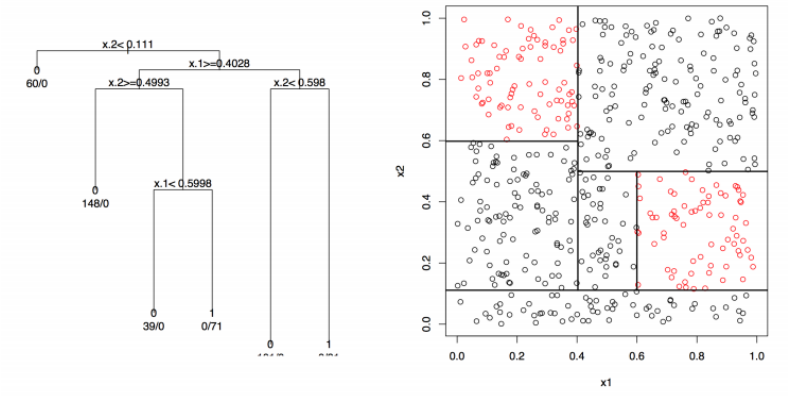


Figure 3.14: *Decision tree and induced partitioning of* $\mathbb{R}^2$ *sample space*

To make out hypothesis $\mathcal{H}_{CT}$ class smaller and simpler, we will focus on disjoint unions of boxes that are obtained by iteratively splitting an existing box into two smaller boxes along one of the axes:

- We start with the whole sample space $\mathbb{R}^d$.
- By selecting some coordinate $i_1 \in [d]$ and some value $t_1 \in \mathbb{R}$ we split R^d into two axis-parallel "boxes" (half-spaces). We obtain:

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

- Next, by focusing of some previously split "box" B_j^s for $s \in \{-,+\}$, we can again select some coordinate $i_{j+1} \in [d]$ and some splitting value $t_{j+1} \in \mathbb{R}^d$ to obtain B_{j+1}^-, B_{j+1}^+. Notice that B_{j+1}^- and B_{j+1}^+ are disjoint, and if following this procedure then by induction they are also disjoint from any other obtained box.

Note that the partitions obtained this way are special - most partitions of $\mathbb{R}^d$ into axis-aligned boxes are not Tree Partitions. Namely, cannot be constructed by such a top-down iterative chopping procedure.

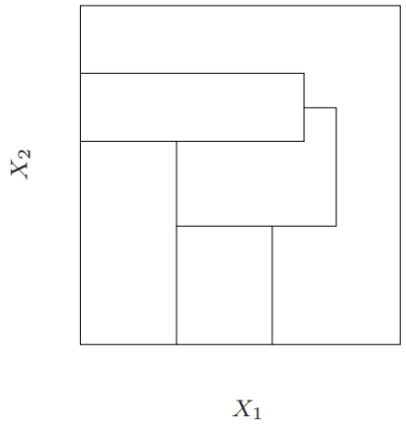


Figure 3.15: *Partitioning $\mathbb{R}^2$ into axis-aligned boxes not describing a tree partition*

3.7.2 Classification & Regression Trees

The hypothesis class $\mathcal{H}_{CT}$ we will consider consists of piecewise-constant functions, that assign a class prediction (1 or 0) to each box in a Tree Partition. Unless we restrict it somehow, the class contains all piecewise-constant functions supported on all Tree Partitions of $\mathbb{R}^d$ (to any number of boxes). Formally, for a Tree Partition $\mathbb{R}^d = \biguplus_{j=1}^N B_j$ of $\mathbb{R}^d$ into N boxes, and label assignments $c_j \in \{0,1\}$ $(j = 1,\ldots,N)$ assigning label c_j to box B_j, the hypothesis $h \in \mathcal{H}_{CT}$ is a function $h : \mathbb{R}^d \to \{0,1\}$ defined by

$$h(\mathbf{x}) = \sum_{j=1}^{N} c_j \mathbb{1}_{B_j}(\mathbf{x})$$

where $\mathbb{1}_{B_j}$ is the indicator function of B_j.

■ **Example 3.6** Consider the following scenario: suppose someone comes into a hispital emergency room. The first step of triage is to determine - fast - whether they are in a life-threatening medical

emergency, or else they can wait in line and receive treatment in a little while. The triage uses a sequence of yes/no questions, such as: Is the patient conscious yes/no?

- If not conscious: classify as **emergency**
- If patient is conscious: is the patient's pulse < 40 beats per minute?
  - If yes (pulse < 40): classify as textbfemergency
  - If no, (pulse ≥ 40): is the patient's pulse > 130 beats per minute?
    - If yes (pulse > 130): is the patient's systolic blood pressure < 80 mmHg?
      - If yes classify as **emergency**
      - If not, is the patient's systolic blood pressure > 140 mmHg? If yes, classify as **emergency**. Otherwise, classify as **no emergency**
    - If not classify as **no emergency**

This is a decision tree that uses three features: conscious (a binary categorical feature), pulse (a numerical feature) and blood pressure (also a numerical feature). See if you can we write a diagram for this decision tree in the shape of a tree, where every node is a question, and every leaf is a decision / classification. The root of the tree is the first question ("conscious yes/no?"). Now observe that every function in our Classification Trees hypothesis class $\mathcal{H}_{CT}$ is equivalent to a decision tree. In the notations of the generic example above, the first question is: "$x_{i_1} > t_1$-yes/no?". If yes, we ask the second question "$x_{i_{2,1}} > t_{2,1}$ - yes/no?". If not, we ask the second question: "$x_{i_{2,2}} > t_{2,2}$ - yes/no?". And so on, until there are no more splits and we have reached a box over which the function in $\mathcal{H}_{CT}$ is constant. If the constant value is 1, we classify / predict class 1. If the constant value is 0, we predict class 0. This is why our hypothesis class is called - classification trees ∎

3.7.3 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 be select $h_s \in \mathcal{H}_{CT}$ based on the training sample S. Suppose we have already obtained a Tree Partition of $\mathbb{R}^d$ is some manner, that consists of N disjoint boxes $\mathbb{R}^d = \biguplus_{j=1}^{N} B_j$. Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ be our taining set and denote the predicted label assigned to box B_j by $\widehat{y}(B_j) \in \{0,1\}$. As such, the number of mis-classification errors that are incurred by the training sample that fall inside B_j is $\sum_{\mathbf{x}_i \in B_j} \mathbb{1}[y_i \neq \widehat{y}(B_j)]$.
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:

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

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

$$\widehat{y}_S(B) = \underset{y \in \{0,1\}}{argmax} P_y^S(B)$$

Applying over the entire Tree Partition, minimizing the empirical risk is achieved by labeling box B_j with $\widehat{y}_S(B_j), \quad j \in [N]$. 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

$$\underset{h\in\mathcal{H}_{CT}}{argmin} L_S(h)$$

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

Looking back at the described procedure 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

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

Selecting A Value For k:
Note that by adding the hyper-parameter k we now have **a family of hypothesis classes**, one for each value of k. The value of k controls the size of the hypothesis class and therefore controls the bias-variance tradeoff:

- 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.

3.7.4 CART Heuristic For Growing Trees

The next challenge is how to find the minimizer of $argmin_{h\in\mathcal{H}_{CT}^k} L_S(h)$ 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 aclosed form expression for the minimizer.
- The Half-space classifier was based on ERM and lead to a simple convex optimization problem.

In contrast to those examples the search space over $\mathcal{H}_{CT}^k$ 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 size[1] .

This is our first encounter with the bitter truth that though the ERM principle is nice, it is often impossible to implement efficiently, especially when the hypothesis class has no Euclidean structure. Therefore, we must resort to defining and using **heuristics**: an approach to solving the optimization

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

problem that does not guaranteed to be optimal, but is still sufficient for finding a solution. While the definition of decision trees, the hypothesis class and prediction assignment to boxes in a tree partition are all canonical, there are several different heuristic approaches to the way we "grow a decision tree', namely to the way we choose an hypothesis in practice.

One common approach, coming out of the statistical learning community, is called **Classification and Regression Trees** (CART). This heuristic cinsists of two stages: **growing** the tree, resulting in a tree that is a little too large, and then **pruning** it to bring it down to the most effective size.
Suppose we have chosen k to be the maximal tree depth. The heuristic of growing a full decision tree with at most k levels will proceed top-down, starting from $\mathbb{R}^d$ and progressively chopping each box into two boxes. A given box is **not chopped** if either:

- The maximum number of levels k has been reached.
- The box has reached a pre-determined minimal number of training samples. At the very least we would not split a box if it consists of only a single training sample.

Chopping is done by finding the **best** coordinate, at the **best** value to chop, and whenever you chop given each half-box the **best** class assignment, in the sense of minimizing misclassification error over the training sample.

Formally, 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:
$$B_t^+ := \left\{ \mathbf{x} \in \mathbb{R}^d | \mathbf{x}_i > t \right\}, \quad B_t^- := \left\{ \mathbf{x} \in \mathbb{R}^d | \mathbf{x}_i \leq t \right\}$$
  - Denote $\widehat{y}_S\left(B_t^{\pm}\right)$ be the class assignment for boxes $B_t^{\pm}$ as defined above, minimizing the empirical misclassification risk in that box.
  - Denote $P^S_{\widehat{y}_S\left(B_t^{\pm}\right)}\left(B_t^{\pm}\right)$ 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:
$$g_i(t) := P^S_{\widehat{y}_S\left(B_t^-\right)}\left(B_t^-\right) + P^S_{\widehat{y}_S\left(B_t^+\right)}\left(B_t^+\right)$$
- Denote t_i to be the **best** chopping point along coordinate i and calculate it for every coordinate:
$$t_i := \underset{t \in \mathbb{R}}{argming_i}(t) \quad i = 1, \ldots, d$$
- Let i^* denote the **best** coordinate along which to chop B:
$$i^* := \underset{i \in [d]}{argming_i}(t_i)$$
- 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 exponentinal in the given input (notice that the hyper-parameter k is also part of the input) we can upper bound this value. As we do not allow empty boxes (and in fact any box with less than some minimal number of samples) the number of nodes (including leaves) in the tree is at most m. We therefore conclude that the time complexity of the CART heuristic is $\mathcal{O}\left(m^2 d\right)$

3.7.5 Pruning a Decision Tree

Add explanation about pruning

4. PAC Theory of Statistical Learning

4.1 Introduction

■ **Example 4.1 Immunotherapy**. We are working on a new medical treatment and we would like to predict for each patient whether the treatment will work for him or not. Available to us is a data set which is shown in the figure below. On the basis of this data we need a no/yes (0 or 1) prediction in order to decide whether or not to apply the treatment. ■

sex	age	Time	Number_o	Area	induration	Result_of_Treatment
1	22	2.25	14	51	50	1
1	15	3	2	900	70	1
1	16	10.5	2	100	25	1
1	27	4.5	9	80	30	1
1	20	8	6	45	8	1
1	15	5	3	84	7	1
1	35	9.75	2	8	6	1
2	28	7.5	4	9	2	1
2	19	6	2	225	8	1
2	32	12	6	35	5	0
2	33	6.25	2	30	3	1
2	17	5.75	12	25	7	1
2	15	1.75	1	49	7	0
2	15	5.5	12	48	7	1
2	16	10	7	143	6	1
2	33	9.25	2	150	8	1
2	26	7.75	6	6	5	1
2	23	7.5	10	43	3	1
2	15	6.5	19	56	7	1
2	26	6.75	2	6	6	1
1	22	1.25	3	47	3	1

Taken from UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]

Figure 4.1: *Immunotherapy Data Set*

Let us list the ingredients of a general problem of this type:

- The individual object, $\mathbf{x}$, that we wish to label. In our case, each such object is a vector $\mathbf{x}$ of d real numbers describing d features of a patient, as shown in the figure.

- The set of all possible $\mathbf{x}$'s is called the ***Domain set,*** $\mathcal{X}$. We can take, for example, $\mathcal{X} = \mathbb{R}^d$ (We could also take a smaller $\mathcal{X}$ since age must be positive and sex can have only two values).
- ***Label set,*** $\mathcal{Y}$: set of possible labels. In our case we can take, for example, $\mathcal{Y} = \{0,1\}$, where 0 corresponds to a recommendation for not giving the treatment and 1 for giving it.
- ***A prediction rule***, $h : \mathcal{X} \to \mathcal{Y}$: used to label future examples. This function is also called a ***predictor***, a ***hypothesis***, or a ***classifier***.

The learner, that is, our algorithm, $\mathcal{A}$, receives an input, called the ***Training data***, which consists of m already-labeled objects $S = ((\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)) \in (\mathcal{X} \times \mathcal{Y})^m$. In our example, $m = 21$ and these objects are the 21 lines in the figure. The learner's output is the prediction rule, $h : \mathcal{X} \to \mathcal{Y}$. So a learner is a map $\mathcal{A} : S \mapsto h$. The learner's task is to produce an h that will be sufficiently correct when labeling future objects, where 'sufficiently' should be clearly defined. If, for a large enough training data, the learner can always come up with such an h, we say that the task is *learnable*.

4.2 A Theoretical framework for learning

The basic questions in machine learning are: Which tasks are learnable? How do we learn learnable tasks? How many training samples do we need in order to learn them? In this chapter we develop the ***PAC theory of learning***, a famous theory that gives, within its definitions and assumptions, a complete answer to these questions, for ***batch supervised learning*** ('batch', meaning that the whole training data set is fed at once into the algorithm and 'supervised' meaning that this set contains the labels, y, and not only the domain set, i.e., the $\mathbf{x}$ values). [Was batch learning defined earlier?].

4.2.1 A Data-generation Model

From now on, we will use the following two assumptions:

- There exists a (deterministic) function f which is the correct classifier, i.e., for every $\mathbf{x}$ there is a single correct label, given by $y = f(\mathbf{x})$. We shall refer to this case as the ***PAC Model***. In contrast, a bit later, we will consider a more general case which we will call the ***Agnostic PAC model*** and in which the *same* $\mathbf{x}$ may appear with *different* labels, even within the same training set.
- All $\mathbf{x}$'s, both those that appear in the training set *and* those in any future test set, are independent and identically distributed (i.i.d) random variables, i.e., they are sampled independently using a distribution $\mathcal{D}$ over the example space $\mathcal{X}$. In particular, this means that the probability, $\mathbb{P}(S)$, of getting the sequence $S = (\mathbf{x}_1, y_1), ..., (\mathbf{x}_m, y_m)$ is given by $\mathbb{P}(S) = \prod_{i=1}^{m} \mathcal{D}(\mathbf{x}_i)$. $\mathbb{P}(S)$ does not depend on the labels y_i's, since, by the previous assumption, these labels are uniquely determined by the $\mathbf{x}_i$'s and f.

Compare the above PAC model to the first learning problem we saw, namely, the Linear Model. There, we also were given m examples, $\mathbf{x}_1, .., \mathbf{x}_m \in \mathcal{X}$ (where we specifically took $\mathcal{X} = \mathbb{R}^d$) but all of them received equal importance, for example, in their contribution to the global error (the loss function). Now, the examples $\mathbf{x}_i$'s are i.i.d samples chosen according to $\mathcal{D}$, i.e., they have different probabilities to appear and therefore may have different weights in the loss function. Another difference is that in the case of the linear model we started with the assumption $y_i = f(\mathbf{x}_i)$ where f was deterministic and linear. Here, we do assume f to be deterministic, but otherwise it may take the form of any possible function from $\mathcal{X}$ to $\mathcal{Y}$. In the linear model we eventually relaxed the deterministic assumption and considered y to be a random function of $\mathbf{x}$ of the form: $y_i = f(x_i) + z_i$. An analogous generalization will take place also here, once we consider the more general, agnostic,

case. Finally we note that, although this chapter will focus on *classifiers*, many principles we will encounter will hold also for linear regression problems.

4.2.2 Generalization Error for Classifiers

For a classification task, we define the ***Generalization Error*** of a hypothesis h as the probability to obtain an $\mathbf{x}$ for which $h(\mathbf{x})$ is different than the correct label $f(\mathbf{x})$:

$$L_{\mathcal{D},f}(h) \equiv \mathbb{P}_{x\sim\mathcal{D}}[h(x) \neq f(x)] \equiv \mathcal{D}(\{x \in \mathcal{X} : h(x) \neq f(x)\})$$

where $\mathcal{D}$ and f are unknowns. The generalization error is also called the ***risk***, or the ***true error***.
Note: One should be critical about an error measure that counts the total number of misclassification errors of a classifier. Recall that there are two kinds of errors a classifier can make: Type-I and Type-II errors, and one is usually much worse than the other. For now, we only note that, assuming $\mathcal{Y} = \{0,1\}$, in the above definition of the true error, $h(x) \neq f(x)$ may refer either to $h(x) = 1, f(x) = 0$ or to $h(x) = 0, f(x) = 1$ where one of these two cases will correspond to a ***False Positive*** event and the other to a ***False negative*** event.

4.2.3 Basic Definitions

Let us summarize the theoretical framework we have developed so far. Our task is to design a learning algorithm (a learner), $\mathcal{A}$. For a given sample size m, $\mathcal{A}$ takes a training sample $S = ((\mathbf{x}_1, y_1), .., (\mathbf{x}_m, y_m))$ and outputs a prediction rule (a hypothesis) $h : \mathcal{X} \to \mathcal{Y}$. For classification problems, $\mathcal{Y} = \{\pm 1\}$, but the framework we develop has a much broader applicability.
We assume that the data points, both in training set and in the test set, are generated by sampling independently $\mathcal{X}$ using a distribution $\mathcal{D}$, which is unknown to us. The labels y are fixed: all occurrences of a particular $\mathbf{x}$ will always be accompanied by the same label, $y = f(x)$, where f is a deterministic function, which, like $\mathcal{D}$, is unknown to us (except for its values at the training data points: $f(x_i) = y_i$, $i = 1, \ldots, m$). Finally, the performance of any candidate rule $h : \mathcal{X} \to \mathcal{Y}$ that our learner may produce, will be evaluated using by how well it will perform on future, unseen, samples - using the expected misclassification rate $L_{\mathcal{D},f}(h) \equiv \mathbb{P}_{x\sim\mathcal{D}}[h(x) \neq f(x)]$.
The following sections will be devoted for a detail understanding of the following important definitions.

Definition 4.2.1 A hypothesis class $\mathcal{H}$ is ***PAC Learnable*** if there exists a function $\tilde{m}_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathcal{A}$ with the following property: For every $\varepsilon, \delta \in (0,1)$ and for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{\pm 1\}$ that satisfies $L_{\mathcal{D},f}(h^*) = 0$ for some $h^* \in \mathcal{H}$, when running the learning algorithm $\mathcal{A}$ on $m \geq \tilde{m}_{\mathcal{H}}(\varepsilon, \delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns an hypothesis $h_S = \mathcal{A}(S)$ such that, with probability of at least $1 - \delta$ (over the choice of the training samples), we have $L_{\mathcal{D},f}(h_S) \leq \varepsilon$. For a PAC learnable hypothesis class, we define the ***Sample Complexity*** of $\mathcal{H}$ for specified ε, δ as the minimal number of samples $\tilde{m}_{\mathcal{H}}(\varepsilon, \delta)$ required for the definition to hold with respect to ε, δ. The Sample Complexity function of $\mathcal{H}$ is denoted $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$.

Definition 4.2.2 Let $\mathcal{H} \subset \{\pm 1\}^{\mathcal{H}}$ be an hypothesis class. For a subset $C \subset \mathcal{X}$ let $\mathcal{H}_C$ be the restriction of $\mathcal{H}$ to C, namely, $\mathcal{H}_C = \{h_C : h \in \mathcal{H}\}$, where for $h : \mathcal{X} \to \mathcal{Y}$, $h_C : C \to \mathcal{Y}$ is the

function such that $h_C(x) = h(x)$ for every $x \in C$. Define the ***VC-dimension*** of $\mathcal{H}$ by

$$VCdim(\mathcal{H}) := \max\{|C| \;\Big|\; C \subset \mathcal{X} \text{ and } |\mathcal{H}_C| = 2^{|C|}\}.$$

Note that $VCdim(\mathcal{H}) \leq \infty$.

4.2.4 The Fundamental Theorem of Statistical Learning

Definitions 4.2.1 and 4.2.2 are interesting since, if we choose PAC learnability as our interpretation of what learnability means, then we have a *well defined necessary and sufficient condition for when it is possible to learn, in terms of the VC-dimension, an exact minimal training sample size we need in order to learn, and a "universal" learner that successfully learns when enough training data is available.* In short, we have a full theory of batch learning - a full theory of when it is possible to generalize from a training sample to new samples, and how to do it. This result is sometimes known as "***The Fundamental Theorem of Statistical Learning***" and states the following (roughly):

- An hypothesis class $\mathcal{H}$ is PAC-learnable *if and only* if $VCdim(\mathcal{H}) < \infty$
- The sample complexity of a hypothesis class with a finite VC-dimension is given approximately by
$$m_H(\varepsilon, \delta) \sim \frac{VCdim(\mathcal{H}) + \log(1/\delta)}{\varepsilon}$$
- The ERM rule achieves this minimum, namely, when learning is possible, ERM learns with a minimial number of examples.

In the following sections, we will inspect the definitions of PAC-learnability and VC-dimension in great detail but in order to do so, let us first present a different perspective of the framework we introduced in Section 4.2.3.

4.2.5 Learning as a Game - first attempt

The framework in Section 4.2.3 can be thought of as a *game* between us and Nature, with a random payoff. The game proceeds as follows. The number of training samples m is given in advanced as a game parameter.

We move first. We choose a learner $\mathcal{A}$ that trains on m samples. This is our strategy. Nature moves second. Nature chooses a distribution $\mathcal{D}$ and a labeling function f. This is Nature's strategy. Importantly, Nature knows the strategy we chose when she chooses her strategy. To calculate the game's payoff, an i.i.d sample S of length m is drawn according to the distribution $\mathcal{D}$ that Nature chose, is labeled according to the function f that Nature chose, and is fed into the learner $\mathcal{A}$ that we chose to obtain the prediction rule $h_S := \mathcal{A}(S)$ (the notation h_S helps us remember that the prediction rule we learn, h_S, strongly depends on the random sample S that we drew). The payoff is $L_{\mathcal{D},f}(h)$. Our goal in the game is to end up with an $L_{\mathcal{D},f}(h)$ which is as small as possible while Nature is an adversary that wants it to be as large as possible. Note that the payoff is random as it depends on the sample S that was drawn. If we're unlucky, and the sample S is "bad", namely, does not represent $\mathcal{D}$ very well, then the rule h_S our learner produces might not generalize well and the random loss (for that draw of S) will be high.

Now you might ask: what's the best strategy for us (how to best design $\mathcal{A}$)? Remember that Nature will know what we chose, and can try to be "cruel", namely, to choose $\mathcal{D}$ and f that our learner did not prepare well for. Also, there's always a chance that we will draw a "lousy" sample S, namely a

misleading sample that will confuse our learner and will cause it to output a rule h_S that will not generalize well. Is there a way to "defend" ourselves against "cruel" strategies $\mathcal{D}, f$ that Nature might play, and against "unlucky" draws of a training sample S? Is there anything at all that can be said in this generality about the problem of learning (i.e., the problem of generalizing from training samples to new samples)?

Definition 4.2.3 Let's define this as ***The Learning Game*** (first version):

- A sample size m is fixed.
- We choose a strategy (a learner) $\mathcal{A}$, that is, a function that matches every sample $S \in (\mathcal{X}, \mathcal{Y})^m$ to a prediction rule $h_s \in \mathcal{Y}^{\mathcal{X}}$ (which, by itself is a function, $h_S : \mathcal{X} \to \mathcal{Y}$)
- Nature knows our strategy, and, after us, chooses a strategy that consists of a probability distribution $\mathcal{D}$ over $\mathcal{X}$, and a label function $f : \mathcal{X} \to \mathcal{Y}$.
- A sample S of size m is drawn according to $\mathcal{D}$ and is labeled according to f
- The sample S is fed into $\mathcal{A}$ to produce a prediction rule $h_S = \mathcal{A}(S)$
- The payoff is $L_{\mathcal{D},f}(h_S)$, namely, the expected fraction of misclassification errors h_S will make on data drawn i.i.d according to $\mathcal{D}$ and labeled according to f. The payoff is *random* since S is random and therefore h_S is random.
- We are going to assume Nature is "cruel" and does her best to win. So we'll look for learners $\mathcal{A}$ for which we can *ensure* that the loss $L_{\mathcal{D},f}(h)$ never exceeds a certain value, *for any* strategy $\mathcal{D}, f$ that Nature might play.

4.3 Probably correct & Approximately correct learners

Definition 4.3.1 Let $0 < \varepsilon < 1$. We say that a learner $\mathcal{A}$ is ***Approximately Correct*** with ***accuracy*** ε, if we are certain (with probability 1) that for any training sample, S, drawn using $\mathcal{D}$, $\mathcal{A}$ will output a prediction rule, h_S, with a loss smaller or equal to ε:

$$D\{S|\ L_{\mathcal{D},f}(h_S) \leq \varepsilon\} = 1$$

Note that the loss is random - it depends on the draw of S - therefore it makes sense to talk about the probability that a learner has a certain loss (a probability that, in the present case, is required to be 1).

Now can ask the following question.
Question: Is there an accuracy parameter, $0 \leq \varepsilon < 1$, for which we can choose a learner, $\mathcal{A}$, that will be approximately correct? In other words, that almost surely (with probability 1), for any sample S drawn according to $\mathcal{D}$, we will have $L_{\mathcal{D},f}(h_S) \leq \varepsilon$?
Answer: **No.** Choose $0 \leq \varepsilon < 1$. The learner cannot hope to produce, regardless of how Nature plays, and with probability 1, a rule h_S with $L_{\mathcal{D},f}(h_S) \leq \varepsilon$. There's always a (small) probability to get a completely "pathological" training sample S that does not represent $\mathcal{D}$ at all. The resulting rule h_S can be wrong on most of $\mathcal{X}$. If S is really bad, h_S can have a loss as high as 1, higher than any ε.

■ **Example 4.2** Choose an accuracy $0 \leq \varepsilon < 1$. Take $\mathcal{X} = \{x_1, x_2\}$. Our learner $\mathcal{A}$ must specify what to predict on a point that was not seen in the training set. Let us consider the case where S contains only x_2's and denote it by S_2, that is, $S_2 \equiv (x_2, f(x_2)), (x_2, f(x_2)), ..., (x_2, f(x_2))$, and assume, without a loss of generality, that in this case our algorithm predicts the label x_1 by $+1$: $h_{S_2}(x_1) = +1$. Now, Nature chooses $\mathcal{D}$ with $\mathcal{D}(\{x_1\}) = \gamma$, $\mathcal{D}(\{x_2\}) = 1 - \gamma$, where γ is a number satisfying $0 < \varepsilon < \gamma < 1$.

Nature also chooses a labeling function f with $f(x_1) = -1 = -h_{S_2}(x_1)$. So in case obtaining S_2 as the training set, the loss, $L_{\mathcal{D},f}(h_{S_2})$ would satisfy

$$L_{\mathcal{D},f}(h_{S_2}) > \gamma > \varepsilon$$

since the probability of drawing x_1 as a test point (on which Nature made sure that our prediction will fail) is γ, and γ was chosen by Nature to be bigger than ε. The probability of obtaining S_2 is $(1-\gamma)^m$, even if very small, is not zero because $\gamma < 1$ and therefore we failed to ensure with probability 1, that the loss will be smaller than ε.
Thus, even on a sample space $\mathcal{X}$ with only two points, for any fixed $\varepsilon > 0$, for every strategy $\mathcal{A}$ we might play, Nature has a strategy $\mathcal{D}, f$, such that with some probability over the choice of training samples of length m, we have $L_{\mathcal{D},f}(h_S) \geq \varepsilon$. As ε was arbitrary, this means that for any arbitrarily "bad" loss ε, Nature can, with non-vanishing probability, cause the game to end with a loss of at least ε. What went wrong? In this example, with probability $(1-\gamma)^m$ (which can be really very tiny if m is large) we get a "lousy" training sample, S_2, that does not represent $\mathcal{D}$ good enough, and does not allow us to generalize. ∎

We therefore conclude that, given any accuracy parameter $0 < \varepsilon < 1$, no learner $\mathcal{A}$ can guarantee, that with probability 1, the loss will not exceed ε: Nature can always find a strategy for which $L_{\mathcal{D},f}(h_S) > \varepsilon$ will have a non-zero probability to occur.
So now that we are aware that "bad" samples may always appear, with non-zero (even if small) probability, we should not aspire for an absolute certainty in achieving a limited loss. We will accept the fact that on these bad, but with limited chance to appear, training samples, the learner $\mathcal{A}$ might "fail completely" (produce h_S with potentially huge loss). We will, from now on, only require a limited certainty, that we will call 'confidence', for having a limited loss. This means that we will demand that the probability of drawing a bad sample will not exceed a certain threshold, that we will denote by δ and that δ will be specified in the initial parameters of the game, together with ε.
Since we no longer demand absolute confidence in having a limited loss, perhaps we can instead require absolute accuracy (zero loss), but with a limited confidence? That is, perhaps we can require that at least for those good training samples (the ones that will have a probability of $1-\delta$ to appear) the loss will vanish?

Definition 4.3.2 Let $0 < \delta < 1$. We say that a learner, $\mathcal{A}$, is ***Probably Correct*** with ***confidence*** δ, if the probability to obtain a training sample, S, for which $\mathcal{A}$ will output a prediction rule, h_S, with a perfect accuracy (zero loss), is larger or equal to $1-\delta$:

$$D\{S|\, L_{\mathcal{D},f}(h_S) = 0\} \geq 1-\delta$$

Note that in definition 4.3.1 we required prefect confidence $\delta = 0$ ("with probability 1"), to have a certain accuracy ($\varepsilon < 1$), while in definition 4.3.2 we require a certain confidence $\delta < 1$ to have a prefect accuracy $\varepsilon = 0$.

Definition 4.3.2 helps us rephrase the question above more formally.
Question: Is there a confidence parameter $0 < \delta < 1$, for which we can choose a learner, $\mathcal{A}$, that will be probably correct? In other words, in the event of a non-pathological sample (the event with probability of at least $1-\delta$) can we achieve perfect accuracy, $L_{\mathcal{D},f}(h_S) = 0$?
Answer: No. Choose $0 < \delta < 1$. The learner cannot hope to produce, with probability of at least $1-\delta$, a rule h_S with $L_{\mathcal{D},f}(h_S) = 0$, independently of how Nature plays. $L_{\mathcal{D},f}(h_S) = 0$ means that

$\mathcal{D}\{h_S(x) = f(x)\} = 1$ (h_S is, with probability 1 with respect to $\mathcal{D}$, always correct). Nature can choose some $x \in \mathcal{X}$ and can play a $\mathcal{D}$ that gives a finite but tiny probability mass to a certain x. Then, with high probability, the sample S will not include x, and therefore, whatever label the learner assigns to $h_S(x)$, it will be a guess and therefore might be incorrect, causing a finite loss.

■ **Example 4.3** Choose a confidence $0 < \delta < 1$. Take $\mathcal{X} = \{x_1, x_2\}$.
As in example 4.2 , we consider the case of the training sample $S_2 \equiv (x_2, f(x_2)), (x_2, f(x_2)), ..., (x_2, f(x_2))$. Once again we assume that in this case our algorithm predicts the label x_1 by $+1$: $h_{S_2}(x_1) = +1$. As in example 4.2, Nature chooses a labeling function f with $f(x_1) = -1 = -h_{S_2}(x_1)$ and a distribution, $\mathcal{D}$, with $\mathcal{D}(\{x_1\}) = \gamma$, $\mathcal{D}(\{x_2\}) = 1 - \gamma$, but now γ is chosen so that $\delta < (1-\gamma)^m$. Note that this requirement means that when m is large, Nature chose the probability of getting x_1, namely, γ, to be close to zero, which means that S_2 is actually a typical ("non-pathological", "good") sample: its probability to appear, $(1-\gamma)^m$ is larger than δ.
As in example 4.2, in case of obtaining S_2 as the training set, the loss, $L_{\mathcal{D},f}(h_{S_2})$ would satisfy

$$L_{\mathcal{D},f}(h_{S_2}) > \gamma$$

and since $\gamma > 0$ that would mean a non-zero loss. Since S_2 has a probability $(1-\gamma)^m$ to appear and since $(1-\gamma)^m > \delta$, the probability of ending up with a non-zero loss is larger than δ. In other words, our learner $\mathcal{A}$ is not a probably correct one. ■

We conclude that for any confidence parameter $0 < \delta < 1$, no learner $\mathcal{A}$ can guarantee, that with probability $1-\delta$, the loss will vanish: Nature can always find a strategy for which $L_{\mathcal{D},f}(h_S) > 0$ will have a probability large than δ to occur. Even a "typical" training sample may miss small areas in $\mathcal{X}$. On these areas the resulting h_S might be wrong. We have to allow h_S to be wrong sometimes (to make some generalization errors) even when the training sample is "typical".
So whatever learner we construct, we can never have an absolute confidence that our loss limited by some fixed ε, neither we can have even a limited-confidence (with probability larger than $1-\delta$, for some fixed δ) in obtaining an absolute accuracy. What we might be able is to have is a limited-confidence that our loss will be limited, which brings us to the following definition.

Definition 4.3.3 Let $\delta > 0$ and $\varepsilon > 0$. We say that a learner, $\mathcal{A}$, is ***Probably Approximately Correct*** with a confidence δ and an accuracy ε if the probability of obtaining a training sample, S, for which $\mathcal{A}$ will output a prediction rule, h_S, with a loss that does not exceeds ε, is larger or equal to $1-\delta$:

$$D\{S|\, L_{\mathcal{D},f}(h_S) = 0\} \geq 1 - \delta.$$

It is important to understand the difference between the *accuracy* ε and the *confidence* δ: *First*, we draw the training sample S at random. The learning algorithm runs on this random input and its prediction is therefore random. If S is by chance "weird" (not representing $\mathcal{D}$ well), the rule h_S produced will be "wrong", namely, it won't generalize well. The number δ is the probability of failure due to a "weird" sample S. *Second*, we test the rule h_S on new data. The new data is also random. $L_{\mathcal{D},f}(h_S)$ is the expected number of errors h_S will make, i.e., its accuracy. The number ε refers to that accuracy.

4.3.1 The game - for Probably Approximately correct learners

Since we can only hope to build a Probably Approximately correct learner, we will update the game definition a little. The sample size m will no longer be fixed. Instead, the accuracy ε and confidence

δ, which our learner is required to achieve, are specified as game parameters. We get to decide on m in our strategy. Note that there is subtle nuance in notation now: when we write $\mathcal{A}$ for the learner, we actually mean a **sequence** of learners - one for each m. It would be better to write $\mathcal{A}_m : (\mathbf{X} \times \mathcal{Y})^m \to \mathcal{Y}^{\mathcal{X}}$, but we won't bother writing $\mathcal{A}_m$ and will continue to use $\mathcal{A}$. So here is our

updated game - The Learning Game (second version): Fix desired accuracy $\varepsilon > 0$ and confidence $\delta > 0$. We play a game against Nature, with random payoff.

- We choose a sample size m and a learner $\mathcal{A} : (\mathcal{X}, \mathcal{Y})^m \to \mathcal{Y}^{\mathcal{X}}$. Both m and $\mathcal{A}$ can depend on (ε, δ).
- Nature knows our strategy, and, after us, chooses strategy that consists of a probability distribution $\mathcal{D}$ over $\mathcal{X}$, and a label function $f : \mathcal{X} \to \mathcal{Y}$. That is, Nature's strategy can depend on (ε, δ) specified, and also on the $m, \mathcal{A}$ we chose.
- A sample S of size m is drawn according to $\mathcal{D}$ and is labeled according to f
- The sample S is fed into $\mathcal{A}$ to produce a prediction rule $h_S = \mathcal{A}(S)$
- The payoff is $L_{\mathcal{D},f}(h_S)$. It is random since S is random and therefore h_S is random.
- We are going to assume Nature is "cruel" and does her best to win. So we'll look for learners $\mathcal{A}$ that have a **guaranteed maximal loss** $L_{\mathcal{D},f}(h)$ **for any** strategy $\mathcal{D}, f$ that Nature might play.
- To determine if we were successful in the game, we play the game many many times (both us and Nature play the same strategies, just the training samples drawn are different). We count and calculate the probability, over the random draws of training samples S, of the event $\{S \sim \mathcal{D}^m \,|\, L_{\mathcal{D},f}(h_S) \leq \varepsilon\}$. If this probability is found to be larger than $1 - \delta$, that is, if the learner $\mathcal{A}$ we chose was Probably Approximately correct with accuracy ε and confidence δ - against Nature's best strategy - **we say that we've been successful (with regards to the parameters ε, δ)**.

If you don't like the game perspective, here our current definition of our learning challenge: The learner doesn't know $\mathcal{D}$ and f. The learner receives an accuracy parameter ε and a confidence parameter δ. It then ask for training data, S, containing $m(\varepsilon, \delta)$ examples (that is, the number of examples can depend on the value of ε and δ, but it can not depend on the unknown $\mathcal{D}$ or f). Finally, the learner should output a hypothesis h_S, that depends only on ε, δ and the training sample S drawn, such that with probability of at least $1 - \delta$ it holds that $L_{\mathcal{D},f}(h) \leq \varepsilon$. That is, the learner should be Probably Approximately correct, with the specified accuracy ε and confidence δ.

Since we can now choose the sample size m, and data costs money, we want m to be as small as possible - as long as we still design a probably approximately correct learner. We sense that there must be some trade-off between ε, δ and $m(\varepsilon, \delta)$ The figure below shows schematically the connection between m,δ, and ε.

4.4 No Free Lunch and Hypothesis Classes

4.4.1 No Free Lunch!

Turns out that, unfortunately, we cannot in general be successful against Nature even in this second version of our game - for any parameters ε, δ... Why? If we know nothing about D and f, and if there are too many possibilities for f, then no matter how large our sample size, m, we can not be confident-enough that we can find an accurate-enough h_S. Let's understand why.

The two examples above (which demonstrated we can't hope to get a learner that enjoys confidence

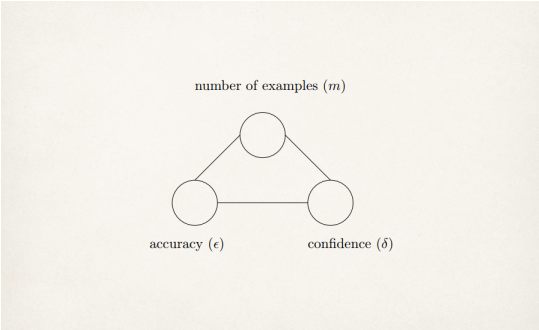


Figure 4.2: *Connection between m,δ, and ε.*

$\delta = 0$ or accuracy $\varepsilon = 0$) only needed a sample space $\mathcal{X}$ with two points. Now let's consider a sample space with a countably infinite number of points. **Example.**

- Suppose that $|\mathcal{X}| = \infty$. Recall that we move first and choose m, and that Nature knows the m we chose. So, let our choice of m be fixed. We must choose what to predict on a point we have not seen in the training sample. Denote our choice by $g(x)$. That is, our learner $\mathcal{A}$ specifies that if a point $x \in \mathcal{X}$ was **not** observed in our training data, then the rule that $\mathcal{A}$ outputs will predict $h_S(x) = g(x)$.
- Now, Nature picks some finite set $C \subset \mathcal{X}$ with $|C| > 2m$, and chooses $\mathcal{D}$ to be uniform over C. For the labeling function f, Nature plays a sinister move, and chooses $f(x) = -g(x)$ for all $x \in \mathcal{X}$. (The opposite of what h_s will predict on unseen points!)
- Now, let S be a training sample. Let $supp(S) \subset C$ denote all points $x \in \mathcal{X}$ observed in the particular training sample S. Since $|supp(S)| \leq m$ and since $\mathcal{D}$ is uniform over C, we have $\mathcal{D}(\{x \in \mathcal{X} \setminus supp(S)\}) \geq 1/2$. In other words, the probability that a new test point drawn according to $\mathcal{D}$ will be unseen in S is at least $1/2$.
- Now, as the game demands, we feed the sample S into $\mathcal{A}$ and obtain $h_S = \mathcal{A}(S)$. What is the loss? Regardless of what h_S predicts on points in the training sample S, since a test point has probability $\geq 1/2$ to be in the unseen part $\mathcal{X} \setminus supp(S)$, with probability at least $1/2$, the rule h_S will predict $h_S(x) = g(x)$ - and will be wrong, since Nature played the function $f(x) = -g(x)$. We conclude that, on the particular training sample S, we must have $L_{\mathcal{D},f}(h_S) \geq 1/2$.
- But this happens for **every** training sample S. So, Nature played a strategy for which, with probability 1 over the choice of training samples (according to $\mathcal{D}$), the game results in loss $L_{\mathcal{D},f}(h_S) \geq 1/2$.
- So, if we were looking to find a learner $\mathcal{A}$ that will be Probably Approximately correct (to some specified ε and δ regardless of Nature's strategy $\mathcal{D}, f$, we find that we cannot. And asking for a larger training sample won't help - if we increase m, Nature will just choose a larger set C and a distribution $\mathcal{D}$ uniform over that larger C.
- What went wrong? Nature could choose **any labeling function at all** f that she wanted, and we tried to learn (to generalize = to accurately predict) f from a sample that was too small for the number of possible functions we needed to choose from. We find we just cannot design a Probably Approximately correct learner if the set of possible labeling functions is "too large".

This is known as a **"No Free Lunch" Theorem**: without assuming anything in advance on f, without an prior information on the labeling function we are trying to learn, we find that learning

is impossible. Equivalently, if the set of possibilities for the labeling function f is too large, then Nature can be cruel and play a function that we can't learn - the larger the sample size we choose, the more complicated the function f that Nature playes. So if the set of possibilities for the labeling function f is too large, learning is impossible (in the sense of the game that we defined). **Important**

Note! The argument given above is in fact wrong (can you find where?). While this was not a proof of a No Free Lunch Theorem, it gives all the crucial bits of intuition, and we will need them later. So we will call it a "proof" in quotes. Here is an actual, formal No Free Lunch theorem. (There are

many theorems that can be called by that name - basically any theorem that shows that without some prior knowledge on the labeling function, when there are "too many" possibilities for f, learning it is impossible.) You can read the proof in the Understanding Machine Learning book - Theorem 5.1, and exercise 3 in section 5.5. (it is not so easy to read and uses Agnostic PAC, a notion we will only get to later.)

Theorem 4.4.1 — No Free Lunch. $\mathcal{X}$ be an infinite sample domain, $|\mathcal{X}| = \infty$. Fix $\varepsilon < 1/2$. There always exists some $\delta > 0$ so that, for every learner $\mathcal{A}$ and training set size m, there exists a distribution $\mathcal{D}(x)$ over $\mathcal{X}$ and a function $f : \mathcal{X} \to \mathcal{Y}$, such that with probability of at least δ over the generation of a training sample S of size m drawn i.i.d from $\mathcal{D}$, we have $L_{\mathcal{D},f}(h_S) \geq \varepsilon$ where $h_S = \mathcal{A}(S)$.

4.4.2 We need hypothesis classes

So, in order to be able to learn, the learner **must** receive enough prior knowledge about the function f. This implies that we should assume that the target f comes from some **hypothesis class**, $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$.

4.4.3 Realizability Assumption.

Suppose that an hypothesis class $\mathcal{H}$ is specified for our game. The **realizable case** is when Nature must play a function $f \in \mathcal{H}$. Actually, we don't care if Nature plays a function f that is not in $\mathcal{H}$ as long as f is $\mathcal{D}$-almost surely identical to a function $h^* \in \mathcal{H}$. (This is because we will never see samples in $\mathcal{X}$ where $f(x) \neq h(x)$ - not in the training and not in the test samples.) So the formal mathematical **Realizability Assumption** is this: Nature plays a function f such that there exists $h^* \in \mathcal{H}$ with $L_{\mathcal{D},f}(h^*) = 0$. The learner is given $\mathcal{H}$ before learning starts, and will only output

$h_S \in \mathcal{H}$. In other words, for a training sample of size m the learner (learning algorithm) is a map $\mathcal{A} : (\mathcal{X} \times \mathcal{Y})^m \to \mathcal{H}$ such that $\mathcal{A} : S \mapsto h \in \mathcal{H}$. (As before, we will continue to abuse notation and write $\mathcal{A}$ instead of $\mathcal{A}_m$, and when we say "the learning algorithm $\mathcal{A}$" we will sometimes mean "a sequence of learners $\{\mathcal{A}_m\}_{m=1}^{\infty}$, one for each possible sample size").

We've just seen that if $|\mathcal{X}| = \infty$ then $\mathcal{H} = \mathcal{Y}^{\mathcal{X}}$ is "too large to learn". And, in a previous section we've seen that the learner gets to specify the training sample size m. The questions that arise now are:

- What are the "small enough" hypothesis classes $\mathcal{H}$ for which we **can** find a Probably Approximately correct learner? And what are the "too large" hypothesis classes $\mathcal{H}$ for which we **cannot**? Can we characterize exactly the hypothesis classes for which it is possible to learn?
- Assume we have a "small enough" $\mathcal{H}$. This means that for every ε, δ we have at least one strategy $m, \mathcal{A}$ such that $\mathcal{A}$ is Probably Approximately correct (with accuracy ε and confidence δ) no matter how Nature plays. This means that for every ε, δ there is a **minimal number**

of training samples: maybe some learners are wasteful and need more training data than others, but for every ε, δ there is the absolutely minimal training sample size that allows us to choose a Probably Approximately correct learner. Can we characterize this minimal function $m_{\mathcal{H}}(\varepsilon, \delta)$? Is there a connection between the "size" of the hypothesis class $\mathcal{H}$ and $m_{\mathcal{H}}(\varepsilon, \delta)$?

- Assume we have a "small enough" $\mathcal{H}$. Can we characterize explicitly a learner $\mathcal{A}$ that always succeeds in learning functions from $\mathcal{H}$? And how many training samples m does $\mathcal{A}$ need to always succeed in learning a function from $\mathcal{H}$ (always be Probably Approximately correct, no matter how Nature plays)? Can we find the **most training-data efficient** learner, namely a learner that can succeed with the minimal number of samples $m_{\mathcal{H}}(\varepsilon, \delta)$ mentioned above?

4.4.4 Updating the game one last time

So here is our final version of the game - (**The Learning Game (third version):** Fix desired accuracy $\varepsilon > 0$ and confidence $\delta > 0$. Fix an hypothesis class $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$. We play a game against Nature, with random payoff.

- We choose a sample size m and a learner $\mathcal{A} : (\mathcal{X}, \mathcal{Y})^m \to \mathcal{H}$. Both m and $\mathcal{A}$ can depend on (ε, δ).
- Nature knows our strategy, and, after us, chooses strategy that consists of a probability distribution $\mathcal{D}$ over $\mathcal{X}$, and a label function **from the specified hypothesis class** $f \in \mathcal{H}$. That is, Nature's strategy can depend on $\varepsilon, \delta, \mathcal{H}$ specified, and also on the $m, \mathcal{A}$ we chose.
- A sample S of size m is drawn according to $\mathcal{D}$ and is labeled according to f
- The sample S is fed into $\mathcal{A}$ to produce a prediction rule $h_S = \mathcal{A}(S)$. Note that $h_S \in \mathcal{H}$.
- The payoff is $L_{\mathcal{D},f}(h_S)$. It is random since S is random and therefore h_S is random.
- We are going to assume Nature is "cruel" and does her best to win. So we'll look for learners $\mathcal{A}$ that have a **guaranteed maximal loss** $L_{\mathcal{D},f}(h)$ **for any** strategy $\mathcal{D}, f$ that Nature might play.
- To determine if we were successful in the game, we play the game many many times (both us and Nature play the same strategies, just the training samples drawn are different). We count and calculate the probability, over the random draws of training samples S, of the event $\{S \sim \mathcal{D}^m \,|\, L_{\mathcal{D},f}(h_S) \leq \varepsilon\}$. If this probability is found to be larger than $1-\delta$, that is, if the learner $\mathcal{A}$ we chose was Probably Approximately correct with accuracy ε and confidence δ - against Nature's best strategy - **we say that we've been successful (with regards to the parameters ε, δ) and hypothesis class $\mathcal{H}$.**

4.4.5 Example: Threshold functions

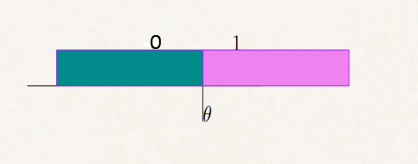
We saw above that if $|\mathcal{X}| = \infty$ and $\mathcal{H}$ is the class of all functions from $\mathcal{X}$ to $\mathcal{Y}$, namely $\mathcal{H} = \mathcal{Y}^{\mathcal{X}}$, we can't be successful in the the third version of the learning game. Well, maybe it's just impossible to learn when $|\mathcal{X}| = \infty$? Fortunately, when the hypothesis class is "small enough", it *is* possible to be successful in the third version of the learning game, which is the reason why learning is possible . In the following example we will have an infinite (in fact uncountably infinite) sample space, and a very simple hypothesis class. We will see that we can be successful in the third version of the game, for any ε, δ specified.

Consider the domain $\mathcal{X} = \mathbb{R}$, i.e., there is only one feature. We work with classification, and use the label set is $\mathcal{Y} = \{0, 1\}$ (instead of $\{+, -\}$). Define the hypothesis class of **Threshold functions:** over $\mathbb{R}$:

$$\mathcal{H}_{th} = \{x \mapsto h_\theta(x) : \theta \in \mathbb{R} \bigcup \pm\infty\}$$

where $h_\theta(x) = 0$ for $x \leq \theta$ and $h_\theta(x) = 1$ for $x > \theta$. (We define $h_\infty(x) = 0$ for all x, and $h_{-\infty}(x) = 1$

for all $x \in \mathbb{R}$).



What's the meaning of this definition? We are classifying points on the real line. Up until a certain unknown point, the points are in class 0. Beyond that point, they are in class 1. Nature chooses the true cutoff threshold θ, and a distribution $\mathcal{D}$ over the real line. We would like to receive a training sample S of labeled points, and successfully predict the label of future examples. In this case, our job is to determine, as accurately as possible, the unknown cutoff θ. (It is recommended to take some time and understand all the details of this example!)
Now that we know our hypothesis class to be $\mathcal{H}_{th}$, we should specify our strategy, which consists of the number of samples m we need, and a learning algorithm $\mathcal{A}$ that will process training sample and produce a decision rule. As before let $S = ((x_1, y_1), ..., (x_m, y_m))$ be the training set. As we will see, our choice of learner will not depend on the ε, δ specified, but our choice of m will certainly depend on them.

Learning algorithm

Let's start with a suggested learning algorithm. The training data may take the form shown in the three figures below, but not the form shown in the fourth figure, which is forbidden because it does not obey the Realizability Assumption (the assumption that Nature chooses $h \in \mathcal{H}$).

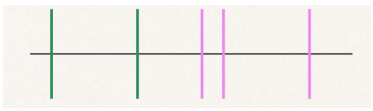


Figure 4.3: *Possible training data for $\mathcal{H}_{th}$*

Figure 4.4: *Possible training data for $\mathcal{H}_{th}$*

Figure 4.5: *Possible training data for $\mathcal{H}_{th}$*

We suggest the following learning algorithm: return hypothesis $h_{\theta_{alg}}(x)$ with

$$\theta_{alg} = \max_{y_i=0} x_i$$

Figure 4.6: *Forbidden training data for $\mathcal{H}_{th}$ - violates the Realizability Assumption*

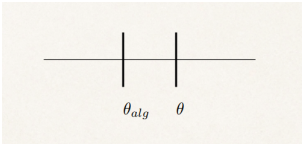
. If $y_i = 1$ for all $i = 1 \ldots m$ then return $\theta_{alg} = -\infty$, namely, return a rule that classifies all points to 1. Similarly, if $y_i = 0$ for all $i = 1 \ldots m$ then return $\theta_{alg} = +\infty$, namely, return a rule that classifies all points to 0.

Number of Samples

Let $0 < \varepsilon, \delta$ be the accuracy and confidence specified in the game. What should be out choice of m? Namely, how many samples are needed to guarantee that the true error is at most ε with probability at least $1 - \delta$?

Claim 4.4.2 Fix $0 < \varepsilon, \delta$. If $m \geq \frac{\log(1/\delta)}{\varepsilon}$ then for any distribution $\mathcal{D}$ over the real line, and any choice of labeling threshold function $f_\theta \in \mathcal{H}_{th}$, with probability of at least $1 - \delta$ (over the choice of training sample S of size m), the loss $L_{\mathcal{D},\theta}(h_{\theta_{alg}})$ of the algorithm for learning threshold functions is at most ε.

Proof: Fix distribution $\mathcal{D}$ over the domain set. Fix correct hypothesis $f_\theta \in \mathcal{H}_{th}$.

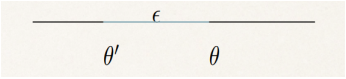


From the properties of the algorithm we know that:

$$\theta_{alg} < \theta$$

Note that the prediction rule produced by the algorithm will be correct for test samples $x < \theta_{alg}$ and $x > \theta$ and is incorrect for $\theta_{alg} < x < \theta$.

If $\mathcal{D}(\{x : -\infty < x < \theta\}) < \varepsilon$ then $\mathcal{D}(\{x : \theta_{alg} < x < \theta\}) < \varepsilon$ and therefore the true error is *always* (that is, with probability 1, no matter what S or m is) smaller than ε and we are done. We will therefore assume that $\mathcal{D}(\{x : -\infty < x < \theta\}) \geq \varepsilon$ and define θ' such that $\mathcal{D}(\{x : \theta' < x < \theta\}) = \varepsilon$.



Note that if there is $(x, y) \in S$ with $\theta' \leq x \leq \theta$ then the true error is at most ε and the probability not to get such a test sample is $(1 - \varepsilon)^m$. Using $1 - \varepsilon \leq e^{-\varepsilon}$, we see that the term $e^{-\varepsilon m}$ would be smaller than δ if $m \geq \frac{\log(1/\delta)}{\varepsilon}$.

4.4.5.1 Threshold functions - conclusion

We saw that, for $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = \{0,1\}$ and $\mathcal{H} = \mathcal{H}_{th}$ (the hypothesis class of threshold functions), we have a strategy (choice of sample size $m = m(\delta,\varepsilon)$ and learning algorithm $\mathcal{A}$) that is **always successful against Nature**, for any values ε,δ specified. In other words, for any ε,δ, our strategy (which depends on δ,ε) specified) is a Probably Approximately correct learner regardless of how Nature plays. We recall that for $\mathcal{X} = \mathbb{R}$, $\mathcal{Y} = 0,1$ and $\mathcal{H} = \mathbb{R} \to \mathbb{R}$ there were values of ε,δ for which we **could not be successful** regardless of how we played (in fact, we could not be successful for any $\varepsilon < 1/2$). So whether we can be successful for any values ε,δ seems to be a property of the hypothesis class we choose. This brings us to the famous definition of a **Probably Approximately Correct (PAC) learnable hypothesis class**.

4.5 PAC learning

Definition 4.5.1

1. A hypothesis class $\mathcal{H}$ is **PAC Learnable** if there exists a function $\tilde{m}_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ and a learning algorithm $\mathcal{A}$ with the following property: For every $\varepsilon,\delta \in (0,1)$ and for every distribution $\mathcal{D}$ over $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{\pm 1\}$ that satisfies $L_{\mathcal{D},f}(h^*) = 0$ for some $h^* \in \mathcal{H}$, when running the learning algorithm $\mathcal{A}$ on $m \geq \tilde{m}_{\mathcal{H}}(\varepsilon,\delta)$ i.i.d. examples generated by $\mathcal{D}$ and labeled by f, the algorithm returns an hypothesis $h_S = \mathcal{A}(S)$ such that, with probability of at least $1-\delta$ (over the choice of the training samples), we have $L_{\mathcal{D},f}(h_S) \leq \varepsilon$.
2. For a PAC learnable hypothesis class, we define the **Sample Complexity** of $\mathcal{H}$ for specified ε,δ as the minimal number of samples $\tilde{m}_{\mathcal{H}}(\varepsilon,\delta)$ required for the definition to hold with respect to ε,δ. The Sample Complexity function of $\mathcal{H}$ is denoted $m_{\mathcal{H}}(0,1)^2 \to \mathbb{N}$.

Note that PAC learnability is only one possible definition of learning that can account for the fundamental limitations on accuracy and confidence. We could, for example, settle for a specific, "good enough", values of δ, and ε instead of requiring that the condition that $m > m(\varepsilon,\delta)$ implies $L_{\mathcal{D},f}(h) \leq \varepsilon$, to hold, for **any** $\delta,\varepsilon < 1$.

4.5.1 Finite hypothesis classes are PAC learnable

In order to understand more about when PAC learning is possible (namely, which hypothesis classes are PAC learnable) let us first consider the case where $\mathcal{H}$ is a *finite* hypothesis class. Finite hypothesis classes can be huge: for example, take $\mathcal{H}$ is all the functions from $\mathcal{X}$ to $\mathcal{Y}$ that can be implemented using a Python program of length at most b, for b fixed and large. Or, take $\mathcal{H}$ to be all the functions from $\mathcal{X}$ to $\mathcal{Y}$ where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite.

4.5.1.1 Empirical Risk Minimization

You might expect that there would be nothing to say in this generality, namely, that in order to design a successful learning algorithm we must pay attention to the details of the specific $\mathcal{X}$ and finite $\mathcal{H}$ at hand. Now comes a big surprise. It turns out that **there is a simple learner that is always successful on finite hypothesis classes** (and on many other hypothesis classes as we will see later).
The idea behind this amazing learning is very simple and natural: try to be as correct as possible on the training data!
Formally, given a training set $S = (x_1,y_1),\ldots,(x_m,y_m)$ we define the **empirical risk** of a candidate

prediction rule $h \in \mathcal{H}$ by

$$L_S(h) = \frac{1}{m}|\{i : h(x_i) \neq y_i\}|\,.$$

Our amazing learning algorithm is simple: on a training sample S, it returns $h \in \mathcal{H}$ that **minimizes the empirical risk** $L_S(h)$**.** In other words,

$$\mathcal{A}_{ERM} : S \mapsto \text{argmin}_{h\in\mathcal{H}} L_S(h)\,.$$

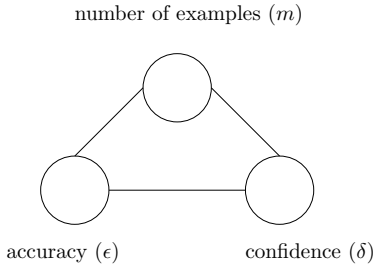
The minimum may not be unique, in which case the algorithm returns one of the minimizers. Our amazing learner is therefore called **Empirical Risk Minimization (ERM)** learner. We give this important learner its own special notation and denote it by $ERM_{\mathcal{H}}$ instead of Ac_{ERM}.
But wait, how do we know that there is a minimum? Note that $L_S(h) \geq 0$, and we are minimizing over a finite class, so there is a minimum. In fact, under the assumption that Nature plays $f \in \mathcal{H}$, we know that for any training sample S, $L_S(f) = 0$ for the particular labeling function that Nature chose. So that the lower bound 0 is achievable. In other words, under our assumption that $f \in \mathcal{H}$, the ERM learner will always return a rule h with $y_i = h(x_i)$ for $i = 1,\ldots,m$. Such a rule is called **Consistent** - it is consistent with the training sample.

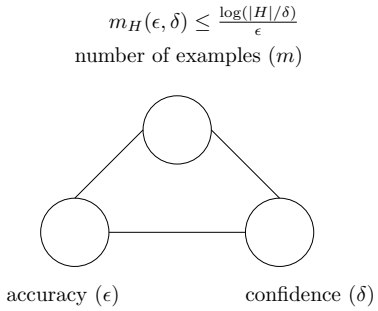
4.5.1.2 Learning Finite Classes

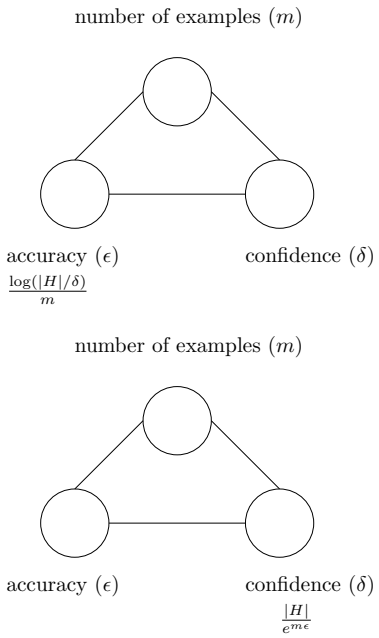
Our main observation concerning finite classes is simple: a **finite** hypothesis class $\mathcal{H} \subset \mathcal{Y}^{\mathcal{X}}$ is PAC-learnable, using the ERM rule, with sample complexity at most $\log(|\mathcal{H}|/\delta)/\varepsilon$.

Theorem 4.5.1 Fix $0 < \varepsilon, \delta < 1$. If $m \geq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$ then for every $\mathcal{D}, f$, with probability of at least $1-\delta$ (over the choice of S of size m), $L_{\mathcal{D},f}(ERM_{\mathcal{H}}(S)) \leq \varepsilon$.

The figures below explain schematically the relation between the accuracy, confidence and the sample size for finite classes, as implied by the above theorem.



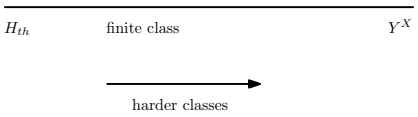




To summarize this section, we have the following: **A finite hypothesis class $\mathcal{H}$ is PAC learnable using the ERM learning algorithm, and has a sample complexity $m_{\mathcal{H}}(\varepsilon,\delta) \leq \log(|\mathcal{H}|/\delta)/\varepsilon$ samples.**
Several natural questions may come to mind.

- Is the bound $m_{\mathcal{H}}(\varepsilon,\delta) \leq \frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$ tight? Can the ERM learner, or an other learner, be Probably Approximately correct (with accuracy ε and confidence δ) using fewer than $\frac{\log(|\mathcal{H}|/\delta)}{\varepsilon}$ samples?
- What happens when noise is present so the y's are not deterministically determined by x?
- What happens when our hypothesis class is infinite?

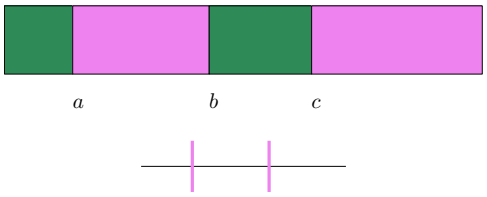
Consider the last question. As we have seen, the class of threshold functions over $\mathbb{R}$, $\mathcal{H}_{th}$, in spite of being infinite, *is* PAC learnable, with sample complexity $m_{\mathcal{H}_{th}}(\varepsilon,\delta) \leq \frac{\log(1/\delta)}{\varepsilon}$, which is obtained by using the $ERM_{\mathcal{H}_{th}}$ learning rule. So $\mathcal{H}_{th}$ appears to be simple to learn. Can we explain why? Somehow, the hypothesis class $\mathcal{H}_{tr}$ is **small** or **simple** and the hypothesis class $\mathcal{Y}^{\mathcal{X}}$ is **large** or **complicated**.
What we need in order to answer the above questions systematically is some sort of a *complexity measure* with which we can order classes by their difficulty along the complexity axis shown in the figure below.



For example, consider the **Two-Intervals** hypothesis class, defined by

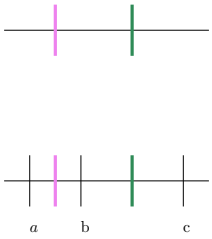
$$\mathcal{X} = \mathbb{R}, \quad \mathcal{H} = \{h_{a,b,c} : a < b < c \in \mathbb{R}\},$$

where $h_{a,b,c}(x) = 1$ if $x \in [a,b]$ or $x \geq c$ and $h_{a,b,c}(x) = 0$ elsewhere.
The following figures demonstrate this point. Suppose we would like to learn with threshold functions the following sample (pink = 1, green = 0)

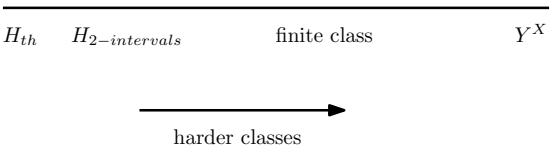


A possible answer would be:

However, if $x_1 < x_2$ and $y_1 = 1, y_2 = 0$, can not be learned by $\mathcal{H}_{th}$, although it can be learned with 2-intervals:



Indeed, we somehow feel that the **2-Interval** class has larger complexity than that of $\mathcal{H}_{th}$ but smaller than that of finite classes.



4.6 VC Dimension

VC-Dimension is the definition of complexity of an hypothesis class we are looking for. This is a **combinatorial measure of complexity** of a function class. What does "combinatorial measure" mean here? It means that VC-dimension is just based on counting stuff, so that $\mathcal{X}$ can be any set, with no additional structure. In particular, the VC-dimension of $\mathcal{H} \subset \{\pm 1\}^{\mathcal{X}}$ is defined even if $\mathcal{X}$ does not need any geometric or algebraic structure. VC-dimension is interesting as it provides a decisive characterization of hypothesis classes that are "simple enough" to learn (in the sense of PAC learnability), versus hypothesis classes that are "too complicated" to learn. It also provides a decisive characterization of $m_{\mathcal{H}}$, the sample complexity of a "simple" hypothesis class $\mathcal{H}$.

4.6.1 Motivation

Suppose we got a training set $S = (x_1, y_1), \ldots, (x_m, y_m)$ and were able to fully explain the labels using a hypothesis from a class $\mathcal{H}$, namely, to find a function $h \in \mathcal{H}$ with empirical risk $L_S(h) = 0$. Suppose that, only to see what will happen, we deliberately corrupt our sample S by changing the labels $\{y_i\}$ - call the corrupt sample S'. Suppose that we **also** succeed in explaining S' using a different

hypothesis from the same class $\mathcal{H}$, namely, find another function $h' \in \mathcal{H}$ with $L_{S'}(h') = 0$. If we can do that, no matter what corrupt labels we choose, it means that something isn't right: how can we hope to generalize based on a training sample S if, regardless of the labels in S, we can find $h \in \mathcal{H}$ with $L_S(h) = 0$? **Definition:** Let $C \subset \mathcal{X}$ be a subset of the sample space and let $h : \mathcal{X} \to \mathcal{Y}$ be some

hypothesis. We define the **restriction** of h to C, denoted $h_C : C \to \mathcal{Y}$, by $h_C(x) = h(x)$, for $x \in C$. **Here is a crucial observation:** Suppose that $\mathcal{H}$ contains **all** functions over some set $C \subset \mathcal{X}$ of size

m, in the sense that $\{h_C \,|\, h \in \mathcal{H}\} = (C \to \mathcal{Y})$. Then we cannot find a Probably Approximately correct learner that uses $m/2$ or fewer training samples.
Why? Go back to the "proof" of the No Free Lunch theorem we saw above. Suppose that indeed there exists a set $C \subset \mathcal{X}$ of size m such that $\{h_C \,|\, h \in \mathcal{H}\} = (C \to \mathcal{Y})$. We play our game against Nature, choose a learner $\mathcal{A}$, and choose a training sample size of $m/2$ or less. Our learner specifies that if it hasn't seen a point $x \in \mathcal{X}$ in the training set, the output rule h_S will predict $g(x)$ for some $g : \mathcal{X} \to \mathcal{Y}$ we specify - we just make sure that the resulting h_S will belong to $\mathcal{H}$. Now, as in the "proof" of the No Free Lunch theorem, Nature plays the distribution $\mathcal{D}$ that is uniform over C, and makes an evil choice to play $f(x) = -g(x)$ for $x \in C$ (Since $\mathcal{D}$ is supported over C, Nature doesn't really care how her f is defined outside C). **This is a legal move for Nature since for every function $\tilde{f} : C \to \mathcal{X}$ there is a hypothesis $f \in \mathcal{H}$ with $f_C(x) = \tilde{f}(x)$, $x \in C$.** Again as in the "proof", our learner fails - regardless of the training sample of size $m/2$, the loss will be $1/2$. We see that, as long

as $\mathcal{H}$ contains **any** set C of size $2m$ with the property that $\{h_C \,|\, h \in \mathcal{H}\} = (C \to \mathcal{Y})$, then we cannot learn with a training sample of size m. It follows that the **maximal size** of such a set C in $\mathcal{H}$ is a **critical quantity**: (i) it gives us a lower bound on $m_{\mathcal{H}}$, the minimal sample size needed, and (ii) if the maximal size is ∞, namely, if for any $m \in \mathbb{N}\mathcal{X}$ contains such as set C with $|C| > m$, the $\mathcal{H}$ is not PAC-learnable!

4.6.2 Formal Definition

Let $C = \{x_1, \ldots, x_{|C|}\} \subset \mathcal{X}$ and let $\mathcal{H}_C$ be the restriction of $\mathcal{H}$ to C, namely, $\mathcal{H}_C = \{h_C : h \in \mathcal{H}\}$. We are working with $\mathcal{Y} = \{\pm 1\}$, so we can represent each h_C as the vector $(h(x_1), \ldots, h(x_{|C|})) \in \{\pm 1\}^{|C|}$. Therefore the total number of possible such vectors is $2^{|C|}$, so that

$$|\mathcal{H}_C| \leq 2^{|C|}\,.$$

We will say that $\mathcal{H}$ **shatters** C if $|\mathcal{H}_C| = 2^{|C|}$.

Definition 4.6.1 The VC dimension of the hypothesis class Hc is defined as

$$VCdim(\mathcal{H}) = \max\{|C| \;:\; \mathcal{H} \;\text{ shatters }\; C\}\,,$$

that is, the VC dimension is the maximal size of a set $C \subset \mathcal{X}$ such that $\{h_C \,|\, h \in \mathcal{H}\} = (C \to \mathcal{Y})$.

We interpret this as the maximal size of a set $C \subset X$ such that $\mathcal{H}$ gives no prior knowledge on label functions restricted to C. According to the above definition, in order to show that $VCdim(\mathcal{H}) = d$

we need to show that:

1. There exists a set C of size d which is shattered by $\mathcal{H}$.
2. Every set C of size k with $k \geq d+1$ is not shattered by $\mathcal{H}$.

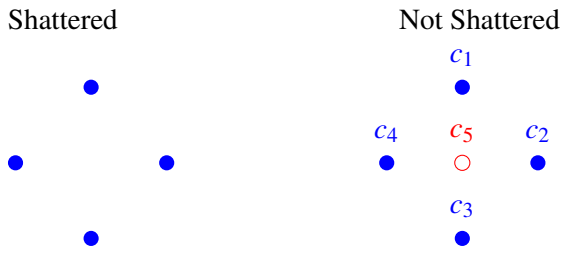
4.6.3 Exercises to help you understand the definition of VC-dimension

Make sure to solve all these exercises. They will help you understand the definition of VC-dimension.

4.6.3.1 Axis aligned rectangles

Consider the **Axis aligned rectangles** hypothesis class over the sample space $\mathcal{X} = \mathbb{R}^2$. We define $\mathcal{H} = \{h_{(a_1,a_2,b_1,b_2)} : a_1 < a_2 \text{ and } b_1 < b_2\}$, where $h_{(a_1,a_2,b_1,b_2)}(x_1,x_2) = 1$ if $x_1 \in [a_1,a_2]$, and $x_2 \in [b_1,b_2]$, and $h_{(a_1,a_2,b_1,b_2)}(x_1,x_2) = 0$ otherwise. (Convince yourself that a function in this hypothesis class is an indicator of a finite open rectangle aligned with the canonical basis of $\mathbb{R}^2$.)

Verify that:



Exercise 4.1 show that no set of 5 points can be shattered by the Axis aligned rectangles class. Hint: note that the 3 points (x_k,y_k), (x_i,y_i), and $(x_{k'},y_{k'})$ can not be shattered if $x_k \le x_i \le x_{k'}$ and $y_k \le y_i \le y_{k'}$. ■

4.6.3.2 Finite classes

Exercise 4.2

- Show that the VC dimension of a finite $\mathcal{H}$ is at most $\log_2(|\mathcal{H}|)$.
- Assume $\mathcal{H}$ is finite. Show that there can be arbitrary gap between $VCdim(\mathcal{H})$ and $\log_2(|\mathcal{H}|)$, namely, construct a finite hypothesis class $\mathcal{H}$ over some sample space $\mathcal{X}$ with $VCdim(\mathcal{H}) = \log_2(|\mathcal{H}|)$ and another finite hypothesis class with $VCdim(\mathcal{H}) = 1$.

■

4.6.3.3 Half-spaces through the origin

Consider the sample space $\mathcal{X} = \mathbb{R}^d$ and the hypothesis class of half-spaces through the origin $\mathcal{H} = \{\mathbf{x} \mapsto sign(\langle \mathbf{w}, \mathbf{x} \rangle) : \mathbf{w} \in \mathbb{R}^d\}$.

Exercise 4.3

- Show that $\{\mathbf{e}_1, , \mathbf{e}_d\}$ is shattered
- Show that any $d+1$ points cannot be shattered (hint: consider the standard basis vectors...)
- What is $VCdim(\mathcal{H})$?

■

5. Ensemble Methods

Previously in the course, we discussed different classification algorithms, each of which implementing a different learning principle for choosing the learning rule $h_s \in \mathcal{H}$. In this chapter we will see how by creating ensembles of classifiers, that is a collection of classifiers where the final prediction is based on all of them, we can improve performance. We will learn abotu the three B's: **Bootstrapping**, **Bagging** and **Boosting** and understand how they give us better control over the **Bias-Variance Trade-off**.

5.1 Bias-Variance Trade-off

Several times in the course so far, we stated - informally - that the "larger" or "more complicated" our chosen hypothesis class, typically our learner will have lower **bias** and higher **variance**. We said informally that bias is part of the generalization error that is incurred by the "best" hypothesis in $\mathcal{H}$. If we think of an unknown labeling function f chosen by nature, then bias measures how well the unknown labeling function f can be decried by the "closest" hypothesis in $\mathcal{H}$. Obviously, the larger $\mathcal{H}$, the more expressive power it has to describe more complicated functions f - hence a lower bias. We said informally that variance is the part of the generalization error that is incurred by the fact that the training sample is random, hence our chosen rule h_s is also random. The larger $\mathcal{H}$ will be, the more freedom our learning algorithm has to "chase" random fluctuations in the training sample, which do not represent the underlying labeling we are trying to learn.

> R The variance part can be further broken down into two parts - one part comes from randomness in the choice of training samples, and another part comes from the measurement noise or noise in the labels. For simpicity we will represent the variance as a single component.

As such, the bias-variance **tradeoff** is that: the more complicated the model, the smaller the bias and the larger the variance. Informally, the generalization error is the sum (or somehow the combination)

of these two. So when we can tune the model complexity (another name for the size / complexity of our hypothesis class) we'll look for the "sweet spot" of a model that not has just the right amount of complexity, not too much and not too little.

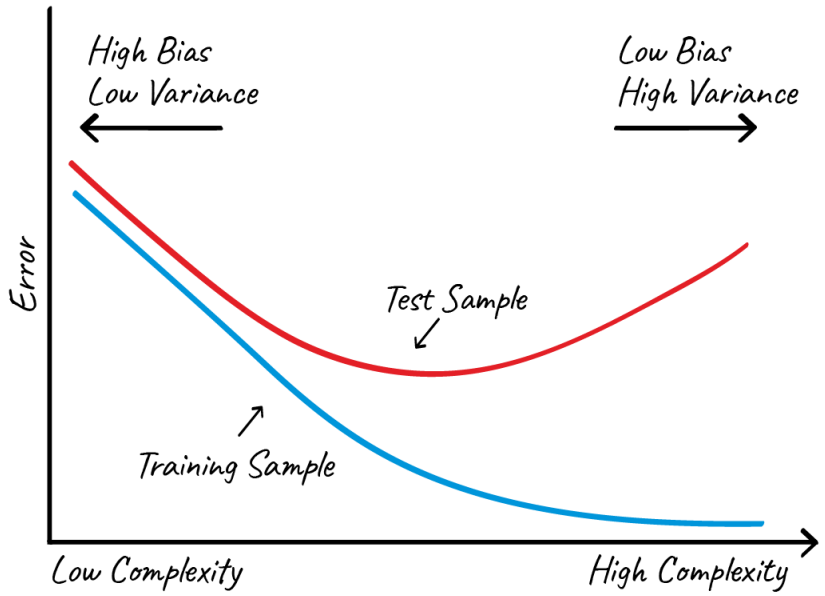


Figure 5.1: The bias-variance tradeoff: *Train- vs. test errors as function of complexity of h*

5.1.1 Generalization Error Decomposition

We have already encountered the generalization error decomposition when discussing linear regression problems (add link-reference to decomposition in regression). There we have shown how we can decompose the MSE into the bias and variance components. Next, let us revisit the decomosition but for a general loss function. Let $h^* = argmin_{h \in \mathcal{H}} L_{\mathcal{D}}(h)$ and $h_S = \mathcal{A}(S)$ be the outout of a learning algorithm, then we can decompose the generalization error of the hypothesis returned by the learner as follows:

$$L_{\mathcal{D}}(h_S) = \underbrace{L_{\mathcal{D}}(h^*)}_{\varepsilon_{approximation}} + \underbrace{L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*)}_{\varepsilon_{estimation}} \tag{5.1}$$

- The **approximation error** is $L_{\mathcal{D}}(h^*)$. Namely, the error of the hypothesis $h \in \mathcal{H}$ achieving the lowest generalization error. This term does not depend at all on our training sample and its size m. It depends only on the selection of $\mathcal{H}$. As we expand the hypothesis class to become richer we might find a better hypothesis for explaining the data. This error is what we already know as the **bias** error, induced by restricting the hypothesis class.
- The **estimation error** is $L_{\mathcal{D}}(h_S) - L_{\mathcal{D}}(h^*)$. Namely, it is the difference between the generalization error achieved by the selected hypothesis and the best hypothesis in $\mathcal{H}$. This term depends on the training set. Finish explaining why this is the variance term

Matan - Add explanation of how to properly create bias-variance curves

5.1.2 Lab: Bias-Variance Via Decision Trees

5.1.3 Lab: Bias-Variance Via Polynomial Fitting

5.2 Ensemble/Committee Methods

> "A collective wisdom of many is likely more accurate than any one." — Aristotle, in *Politics*, circa 300BC

Before diving into specific ensemble methods, let us analyze some mathematical properties of a committee based decision. This will give us insights into how committee based methods manage to inprove generalization. Consider a committee of T members, which has to make a "yes"/"no" decision. Each member casts a vote, which with probability p being correct and probability $1-p$ being wrong. Let's assume for simplicity that all members are "equally wise", so that p is the same for all members. After all members vote, the committee's decision is simply the majority vote. For this setup we can ask questions such as:

- What is the probability of the committee deciding the right decision?
- What would a typical decision be? and how consistent is it?
- How does the number of members in the committee influence the measures above?
- If committee members are not independent from one another, and influence each other's decisions, how does it influence the measures above?

Exercise 5.1 Let $X_1,\ldots,X_T \overset{iid}{\sim} Ber(p)$ taking values of $\{\pm 1\}$. What is the probability of the committee deciding right? ■

Proof. As the committee decides by a majority vote then the probability of the committee deciding right is the same as the probability of having more members deciding right than members deciding wrong:

$$\mathbb{P}(\text{Committee decides right}) = \mathbb{P}(|\text{Decided right}| > |\text{Decided wrong}|)$$

As each random variable takes a value in $\{\pm\}$ we could express the collective vote as $X = sign\left(\sum_{i=1}^{T} X_i\right)$. If the committee decided right, then ther are more members that voted right and $\sum_{i=1}^{T} X_i > 0$ which means that $X = 1$. On the otherhand, if the committee decided wrong, then more members voted wrong and $\sum_{i=1}^{T} X_i <= 0$ which means that $X = -1$.
So, we conclude that:

$$\mathbb{P}(\text{Committee decides right}) = \mathbb{P}(X > 0)$$

■

Exercise 5.2 Let $X_1,\ldots,X_T \overset{iid}{\sim} Ber(p)$ taking values of $\{\pm 1\}$ with $p > 0.5$. Bound below the probability of the committee being correct. ■

Proof. W.l.o.g let us assume that the correct answer is $+1$ and denote $X = \sum_{i=1}^{T} X_i$. We are therefore interested in bounding from below $\mathbb{P}(X > 0)$, which we will achieve by bounding below $\mathbb{P}(X \leq 0)$. Notice that for any $a > 0$ it holds that:

$$\mathbb{P}(X \leq 0) = \mathbb{P}(-aX \geq 0) = \mathbb{P}\left(e^{-aX} \leq e^{0}\right)$$

Now, using Markov's inequality

$$\mathbb{P}(X \leq 0) \leq \mathbb{E}\left[e^{-aX}\right] = \mathbb{E}\left[e^{-a\sum_i X_i}\right] \overset{iid}{=} \mathbb{E}\left[e^{-aX_1}\right]^T$$

Notice that as $X_1 \sim Ber(p)$ over $\{\pm 1\}$it holds that:

$$\mathbb{E}\left[e^{-aX_1}\right] = pe^{-a} + (1-p)e^{a} \leq e^{a-p+pe^{-2a}}$$

where the last inequality is because $1+x \leq e^x$. Next, we use the inequality $x\ln(x) \geq \frac{x^2}{2} - \frac{1}{2} \quad x \in (x)$. For a selection of $a = \frac{1}{2}ln(2p)$ which is positive for $p > 0.5$ we get that:

$$\begin{aligned}\mathbb{P}(X \leq 0) &\leq \mathbb{E}\left[e^{-aX_1}\right]^T \leq \exp\left(T\left(a-p+pe^{-2a}\right)\right) = e^{T\left(\frac{1}{2}ln(2p)-p+\frac{1}{2}\right)} = e^{Tp\left(-\frac{1}{2p}ln\left(\frac{1}{2p}\right)-1+\frac{1}{2p}\right)} \\ &\leq e^{Tp\left(\frac{1}{2}-\frac{1}{2(2p)^2}-1+\frac{1}{2p}\right)} = e^{-\frac{Tp}{2}\left(\frac{1}{4p^2}-\frac{1}{p}+1\right)} = e^{-\frac{Tp}{2}\left(\frac{1}{2p}-1\right)^2} = e^{-\frac{T}{2p}\left(p-\frac{1}{2}\right)^2}\end{aligned}$$

Finally, we conclude that:

$$\mathbb{P}(X > 0) = 1 - \mathbb{P}(X \leq 0) \geq 1 - e^{\left(-\frac{T}{2p}\left(p-\frac{1}{2}\right)^2\right)}$$

■

The above bound teaches us that for a committee of "non-dump" members (i.e. $p > 1/2$) the probability of being wrong decays with a rate of $\mathcal{O}\left(\frac{1}{e^T}\right)$.Relating this to our learning scheme, it means that the confidence, $1-\delta$, in our prediction increases dramatically as T increases. Next, let us calculate what is a typical decision ($\mathbb{E}(X)$) and how consistent is it ($Var(X)$)?

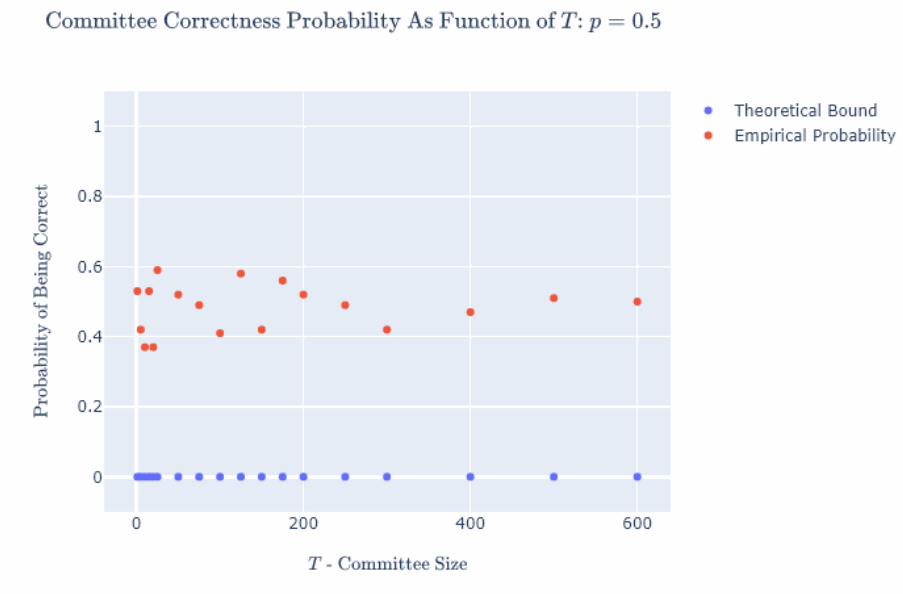


Figure 5.2: Committee Decision - Correctness Probability: *Theoretical bounds and empirical results as function of T, p*

5.2.1 Uncorrelated Predictors

Exercise 5.3 Let $X_1,\dots,X_T \overset{iid}{\sim} Ber(p)$ taking values of $\{\pm 1\}$ with $p > 0.5$. What is the expectation and variance of $X = \frac{1}{T}\sum_{i=1}^{T} X_i$? ■

Proof. We begin with calculating the expectation and variance of each committee member:

$$\begin{aligned}\mathbb{E}[X_i] &= 1\cdot\mathbb{P}(X_i=1)+(-1)\cdot\mathbb{P}(X_i=-1)\\ &= 2p-1\end{aligned}$$

$$\begin{aligned}Var(X_i) &= \mathbb{E}\left[(X_i-\mathbb{E}[X_i])^2\right]\\ &= p(1-(2p-1))^2+(1-p)(-1-(2p-1))^2\\ &= 4p(1-p)^2+4p^2(1-p)\\ &= 4p(1-p)\end{aligned}$$

Then, for X:

$$\begin{aligned}\mathbb{E}[X] &= \tfrac{1}{T}\textstyle\sum_i \mathbb{E}[X_i] = 2p-1\\ Var(X) &= \tfrac{1}{T^2}Var\left(\textstyle\sum_i X_i\right) \overset{iid}{=} \tfrac{1}{T^2}\textstyle\sum_i Var(X_i) = \tfrac{4}{T}p(1-p)\end{aligned}$$

■

Therefore, when using a committee of independent members the expectation of decision remains the same while decreasing the variance at a rate of $\mathcal{O}\left(\frac{1}{T}\right)$. In other word, we are able to keep the same accuracy while increasing the confidence.
Add results of simulations

5.2.2 Correlated Predictors

In practice, however, committee members rarely vote independently. So let us assume that each two members are correlated with equal correlation $\rho \in [0,1]$.

Exercise 5.4 Let $X_1,\dots,X_T$ be a set of identically-distributed real-valued random variables such that: $Var(X_i) = \sigma^2$ and $corr(X_i,X_j) = \rho$, $i \neq j$. What is the variance of $X = \frac{1}{T}\sum_i X_i$? ■

Proof. As X is the average of T identically distributed random variables:

$$Var(X) = Var\left(\frac{1}{T}\sum_i X_i\right) = \frac{1}{T^2}\left[\sum_i Var(X_i) + 2\sum_{i<j} Cov(X_i,X_j)\right]$$

Recall that the correlation between two random variables is defined as:

$$corr(A,B) := \frac{Cov(A,B)}{\sqrt{Var(A)Var(B)}}$$

and therefore for any $i \neq j$:

$$Cov(X_i,X_j) = corr(X_i,X_j)\sqrt{Var(X_i)Var(X_j)} = \rho\sigma^2$$

Plugging this back into the variance:

$$Var(X) = \frac{1}{T^2}\left[T\sigma^2 + 2\binom{T}{2}\rho\sigma^2\right] = \frac{\sigma^2}{T} + \left(1-\frac{1}{T}\right)\rho\sigma^2 = \rho\sigma^2 + \frac{1}{T}(1-\rho)\sigma^2$$

■

Add mathematical analysis of what does this mean
Add simulation results - graph of variance as function of correlation + headmap of dependence on T
All together, we have seen that given a committee of members that decides by majority vote, where decisions of members are correlated with correlation ρ and each member is correct with probability p:

- If $p > 0.5$ the committee's decision improves with T in two ways: higher probability of being right, and its decision will be more consistent.
- If $\rho > 0$ then increasing T will increase the probability of the committee's decision being right up to a certain point,

5.2.3 Committee Methods In Machine Learning

Going back to machine learning, suppose we have T training samples $S_1, \ldots, S_T$ of size m chosen independently from $\mathcal{X}$ according to some distribution $\mathcal{D}$. Let $\mathcal{A}$ be a learning algorithm and train it on each of the training samples, to obtain $h_{S_1}, \ldots, h_{S_T}$. Let us consider $h_{S_t}(x)$ for some $x \in \mathcal{X}$. As $S_t \overset{iid}{\sim} \mathcal{D}^m$ we can think of the training set as a random variable. This means that the also h_{S_t} obtained by training $\mathcal{A}$ over S_t is a random variable. Lastly, it means that we can think of the prediction $h_{S_t}(x)$ as a random variable, which has some distribution. In addition, notice that as the training samples are chosen independently, predictions of different h_{S_t} over x are also independent random variables. So, if we use $h_{S_1}, \ldots, h_{S_T}$ in a committee we have the situation described above. The generalization loss will improve with T, tending to 0 as $T \to \infty$. In addition, the variance of the prediction will decrease as $1/T$.

However, in batch learning we don't have T training samples, but rather just one, so how would we accuire such different hypotheses? We cannot train $\mathcal{A}$ over S multiple times as we will get identical predictions, which are perfectly correlated. Instead, what we would like to do is to create T training samples from the original one. If we could mimic fresh independent draws of new traning samples of size m according to $\mathcal{D}$.

Definition 5.2.1 — Committee Methods. Let $\mathcal{A}$ be some learner predicting labels in $\{\pm\}$. A committee method over $\mathcal{A}$ is the function:

$$h(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right)$$

That is, in committee methods (or ensembles) we take an existing "base" learner and apply it ot a sequence of T training samples. For the remaining of this chapter we will introduce two very different ideas for building the committee member rules.

5.3 Bagging

5.3.1 Bootstrapping

For the first committee method we begin with introducing a concept from statistics: **Bootstrapping**. It is any test or metric that by repeated random sampling **with replacement** of a sample is able to more accurately infer parameters of the sample distribution.
general bootstrap example for some parameter estimation

So, given a training sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ we are going to construct a new training sample, called a *bootstrap sample* S^{*1}. We sample m times from S **with replacement** and denote this "new" sample by:

$$S^{*1} = \left\{\left(\mathbf{x}_i^{*1}, y_i^{*1}\right)\right\}_{i=1}^m$$

Of course, since we sampled from S with replacements, there might be repeated samples in S^{*1}, even if S itself had no repeated samples. Now we can repeat this process B times, obtaining B training samples, each of length m: $S^{*1}, \ldots, S^{*B}$. The samples in the b-th training sample will be denoted

$$S^{*b} = \left\{\left(\mathbf{x}_i^{*b}, y_i^{*b}\right)\right\}_{i=1}^m$$

Using the newly created bootstrap samples we can now train our base learner $\mathcal{A}$ over each one separatly, obtain B prediction rules and form an ensemble. But so how is it that bootstrap actually works? Assume for a moment that samples in our learning problem are i.i.d samples from an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. We are hoping that each Bootstrap from S somehow behaves like a fresh iid sample from $\mathcal{D}$ itself. Given a training sample S (assume for simplicity that all the points of S are distinct) let's define the **empirical distribution** $\widehat{\mathcal{D}}_S$ induced by S on $\mathcal{X} \times \mathcal{Y}$ as the following probability distribution on $\mathcal{X} \times \mathcal{Y}$: for a subset $C \subset \mathcal{X} \times \mathcal{Y}$, define:

$$\widehat{\mathcal{D}}_S((X,Y) = (x,y)) := \begin{cases} \frac{1}{m} & (x,y) \in S \\ 0 & (x,y) \notin S \end{cases}$$

or equivalently, for any $C \subset \mathcal{X} \times \mathcal{Y}$:

$$\widehat{\mathcal{D}}_S(C) := \frac{|C \cap S|}{m}$$

Observe that this is equivalent to putting a probability mass of $1/m$ on each of the points of S, and zero mass on all other points in $\mathcal{X} \times \mathcal{Y}$. **Now observe that a bootstrap sample S^{*b} is just an iif draw of m points** from the empirical distribution $\widehat{\mathcal{D}}_S$ induced by the one training sample we have, S. As m grows, namely as S becomes larger, the empirical distribution $\widehat{\mathcal{D}}_S$ converges in distribution to $\mathcal{D}$. The idea behind the bootstrap is that, if $\widehat{\mathcal{D}}_S$ is not so different from $\mathcal{D}$, then m iid draws from $\widehat{\mathcal{D}}_S$ is a good approximation to m iid draws from $\mathcal{D}$. One way to see the convergence of the empirical distribution to the underlying distribution is on the real line:
Have code for simulation!

Exercise 5.5 In simulation, take $\mathcal{F}$ to be, $\mathcal{N}(0,1)$. For each value $m = 10, 100, 1000$, draw a sample and plot its empirical CDF - and overlay the CDF of $\mathcal{N}(0,1)$. Recall that on the real line, convergence in distribution is equivalent to convergence of the CDFs to a limiting CDF at the continuity points of the limiting CDF. ■

5.3.2 Bootstrapping for Bagging

The idea of Bootstrap samples can be used whenever we would like to create new artificial samples from our only training sample S. It has many uses throughout machine learning, statistics and data science. **Bagging** is a nickname for a straightforward use of the Boostrap in machine learning, to improve accuracy of an existing supervised machine learning algorithm.

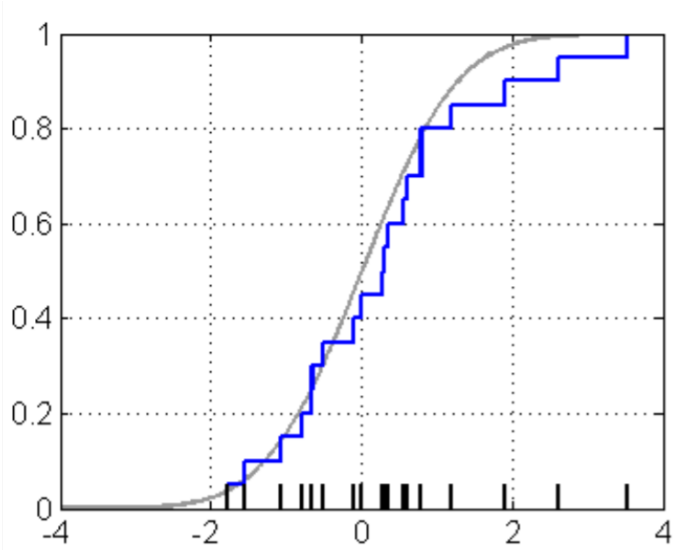


Figure 5.3: *CDF of a probability distribution on the real line, and empirical CDF of an i.i.d sample from that distribution. Black lines on the horizontal axis show the random sample.*

We start with a "base" learning algorithm $\mathcal{A}$ and a training sample S. We choose T (later discussed how) and form T bootstrap training samples, $S^{*1},\ldots,S^{*T}$, each of size m. We then train our learner **separately** on each of the T bootstrap training samples. We form the committee $h_{S^{*1}},\ldots,h_{S^{*T}}$ and store all T trained models. When we need to classify a new test sample $x \in \mathcal{X}$, we run x through all the rules and classify using the majority vote of the committee,

$$h_{bag}(x) := sign\left(\sum_{t=1}^{T} h_{S^{*t}}(x)\right)$$

For example, if we run Bagging on top of the Decision Tree classifier, we'll obtain a committee of decision trees: Add graph+code for num. Bootstraps vs Test error

Note that our learner $\mathcal{A}$ must know how to handle repeated samples. We may have them anyway in S, but running on a bootstrap sample we are sure to have them. Some learning algorithms suffer when there are repeated samples - as they cause numerical problems (for example, linear and logistic regression), while for others it isn't a problem (for example, decision trees and k-NN).

5.3.3 Bagging Reduces Variance

We saw that a committee majority vote reduces variance, but as seen in Figure 5.3 only to a certain degree. The amount of variance reduced is determined by the correlation between committee members. So, we can expect bagging to reduce variance as T increases (and therefore to reduce the generalization error) but only proportionate to the correlation between the bagged prediction rules.

5.3.4 Random Forests Bagging and De-correlating Decision Trees

So, bagging can be improved by somehow de-correlating the bagged prediction rules. How do we de-correlate the committee members - namely, cause their predictions somehow to be less correlated?

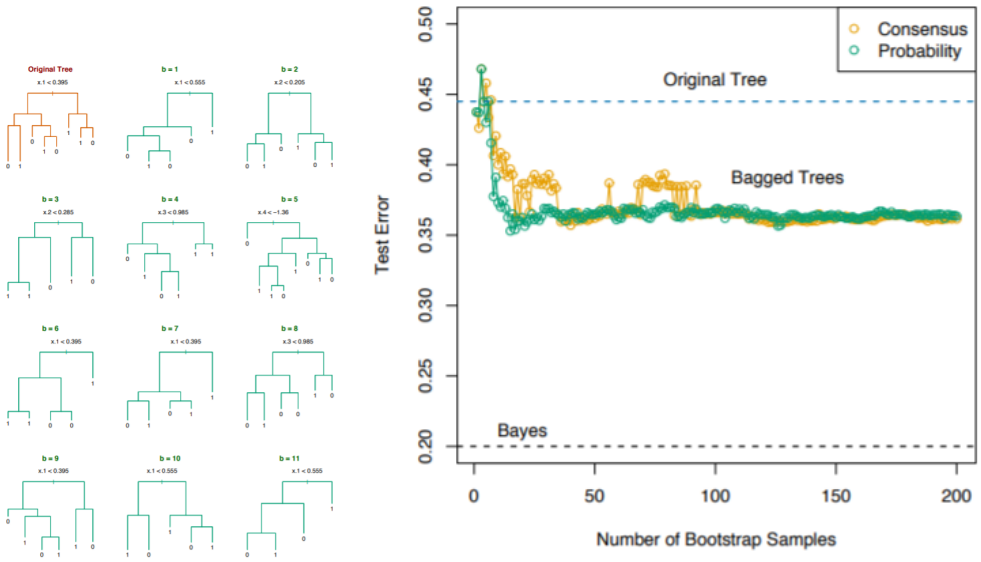


Figure 5.4: *Collection of Bagged Decision Trees. (Source: ESL)*

One way to do this is by handicapping (restricting) each learner a little, in a random way, and hope that the performance gain (in bagging them) due to de-correlation is more than the performance loss to each learner by handicapping. The most well know example of this principle is **Random Forests**

Recall the Decision Tree classification algorithm over $\mathcal{X} = \mathbb{R}^d$. We have a training sample S with m points. The Random Forest classifier is obtained by using Bagging on top of the Decision Tree algorithm, **with an important twist** for de-correlation: the algorithm has a tuning parameter $k \leq d$. When growing each decision tree, in each split, we choose k out of the d coordinate uniformly at random, and only choose the split among these k coordinates. Formally:
add styled pseudo code for random forests
This de-correlation trick works: pretty much on every classification problem you'll work on, you'll observe something like the next plot: Bagging trees is much better than a single tree, and Random Forest (Bagging with the de-correlation trick) is better than just Bagging trees.
Own code for graph without gradient boosting + update caption

5.3.4.1 Bagging - Discussion Points

- **Can Bagging harm our prediction?** Always remember that a committee of fools (a committee where each member has probability $p < 0.5$ to make the right decision) makes worse decisions than a single member. So, when our base learner is so poor that its generalization loss is less than 0.5 we shouldn't use Bagging.
- **What are the disadvantages of Bagging?** As we train not one but T models we need to train all T of them. This directly increases time complexity by a factor proportionate to T. In addition, for predictions, as we need all T models we must store all T of them. Lastly, we loose interpretability as it is much harder to understand why the committee made the decision. We need to understand the decision of each of the T members.
- **Parallelizem** From the computational perspective, it is important to note that Bagging in general (and Random Forests in particular) is **embarrassingly parallelizable**. When training

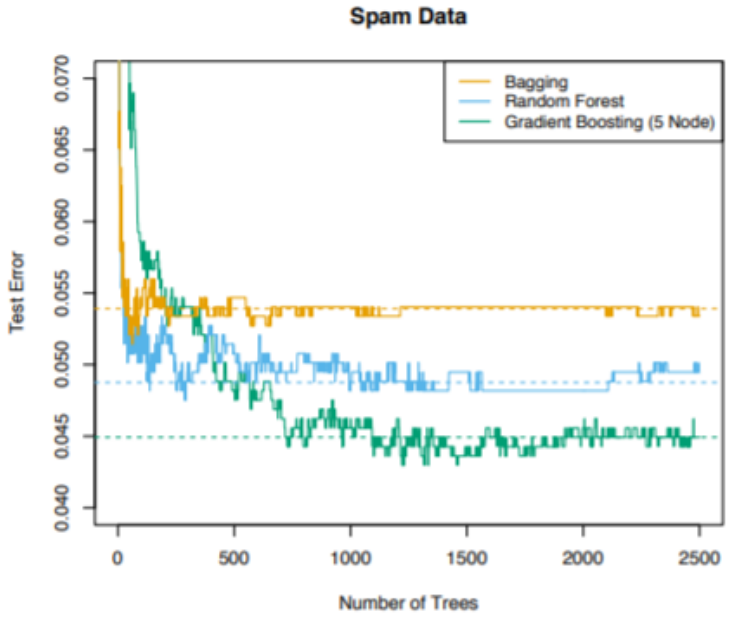


Figure 5.5: *Test error of simple Bagging of decision trees (no de-correlation), Random Forests, and Gradient Boosting of Trees. (Source: ESL)*

a Bagging model with T committee members, we can use T machines in parallel, each using its own random seed to select Bootstrap samples (and random splits, in Random Forest). The machines do not need to interact; when each machine is done, it returns the committee member h_t to the master node.

- **Predicted Class Probabilities:** Can we use the **proportion** of the committee members who voted $+1$ as a predicted class probability? Estimated class probabilities are estimates of $\mathbb{P}\{Y = +1, | X = x\}$. The proportion of members who voted $+1$ estimates $\mathbb{P}\{h_S(x) = +1\}$, which is a different quantity.

5.4 Boosting

Bootstrap was magic of the following kind: we take a single training sample S and turn it into many training samples. Bagging uses this magic by training a model over these "new" training samples, and averaging the result to reduce the variance and hence the generalization error. Boosting is magic of a different kind. In Boosting we take a "weak" learning algorithm, an algorithm with better-than-random but possibly not so good accuracy (i.e. generalization error) and **boost** it using a clever committee method to obtain a learning algorithm with good accuracy.

The core idea of Boosting is a completely different idea for creating a committee of prediction rule from a base learning algorithm $\mathcal{A}$ and a single training sample S. In Bagging, we "pretended" to have fresh training samples $S_1, \ldots, S_T$, and each committee member trained on a different sample. In Boosting, we go even further and "pretend" to have **different underlying distributions** $\mathcal{D}$ from which the training sample is drawn. More specifically, in Boosting each committee member h_t is the result of running $\mathcal{A}$ against a training sample S_t that mimics an i.i.d sample of size m from a **different distribution** $\mathcal{D}^t$. Whereas in Bagging each committee member is trained independently of

all other members, in Boosting the committee members are trained sequentially, one after the other, and each is an improvement, in some sense, on the previous one.
The clever idea behind Boosting is that after we finish training h_t, based on the distribution $\mathcal{D}^t$, we update the distribution in a way that **increases the distribution at training samples where h_t was wrong**. This way, h_{t+1} will try very hard not to be wrong on those particular samples, and so on.
Add Animation of Boosting iterations progress
But first, we have to understand what is meant by "running $\mathcal{A}$ against the training sample S with distribution $\mathcal{D}^t$". One way to interpret this is to take a **weighted Bootstrap** sample from S, where the probability of selecting $(x,y) \in S$ is proportional to $\mathcal{D}^t(x,y)$. A simpler way to interpret this is as follows. If $\mathcal{A}$ uses the ERM principle, say for standard misclassification ($0-1$ loss), namely, looking to minimize the empirical risk,

$$L_S(h) = \sum_{i=1}^{m} \mathbb{1}\left[y_i \neq h(\mathbf{x}_i)\right]$$

then we can use S itself (and not any bootstrap sample) and have the base learner minimize the **weighted** empirical risk

$$L_{S,\mathcal{D}^t}(h) = \sum_{i=1}^{m} \mathcal{D}_i^t \mathbb{1}\left[y_i \neq h(\mathbf{x}_i)\right]$$

where for each $(\mathbf{x}_i, y_i) \in S$ we write $\mathcal{D}_i^t := \mathcal{D}^t(\mathbf{x}_i, y_i)$, so that $\sum_{i=1}^m \mathcal{D}_i^t = 1$.

Observe that these two interpretations are equivalent in expectation. Indeed, the expected number of times for a sample $(\mathbf{x}_i, y_i)$ to appear in the weighted Bootstrap sample is $\mathcal{D}_i^t$, and so it would (in expectation) appear $\mathcal{D}_i^t$ times in the empirical risk sum.

Note that we usually prefer second option (using weighted empirical risk) to the first option (using weighted bootstrap). It's more computationally efficient, and does not require worrying about repeated samples. However, the first option (using weighted bootstrap) is always available. The second option (using weighted empirical risk) is not always possible, and is implemented ad-hoc for the particular base learner we are boosting.
add solution? + fix bullets

Exercise 5.6 To help you understand this point, describe how you would implement a Decision Tree with each of the two methods:

- Using weighted empirical risk: How would you change the Decision Tree algorithm we've seen (CART) to work with a given weight vector $\mathcal{D}^t$ over the training sample S? (Hint: what is the best splitting now that we have weights?)
- Using weighted Bootstrap: How would you How would you change the Decision Tree algorithm we've seen (CART) to work with a given weight vector without changing the splitting algorithm, namely, by giving the algorithm a different training sample selected by weighted Bootstrap? Will the algorithm work with repeated samples?

■

5.4.1 AdaBoost Algorithm

The original Boosting meta-algorithm is known as **Ada**ptive Boosting: Write pseudo down instead of image

Input: training set $S = (x_1, y_1), ..., (x_m, y_m)$, weak learner WL, number of rounds T.
Initialize: initialize $D^1 = (\frac{1}{m}, ..., \frac{1}{m})$
For t **in** $1, ..., T$**:**

- Find weak learner $h_t(x)$ that minimizes ϵ_t, the sum of weights of misclassified points $\epsilon_t = \sum_{h_t(x_i) \neq y_i} D_i^t = \sum_{i=1}^m D_i^t \mathbb{I}(h_t(x_i) \neq y_i)$
- Set $w_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) = \frac{1}{2} \ln\left(\frac{1}{\epsilon_t} - 1\right)$
- Update weights: $\hat{D}_i^{t+1} = D_i^t e^{-y_i w_t h_t(x_i)}$ for all $i \in [m]$.
- Renormalize weights, such that $\sum_i D_i^{t+1} = 1$, by setting: $D_i^{t+1} = \frac{\hat{D}_i^{t+1}}{\sum_i \hat{D}_i^{t+1}}$.

Output: the hypothesis $h_s(x) = sign(\sum_{i=1}^T w_t h_t(x))$

The idea is simple: from iteration t to iteration $t+1$, we want to **increase** the weights of samples misclassified by h_t (where $y_i h_t(\mathbf{x}_i) = -1$) and **decrease** the weights of samples correctly classified by h_t. We want to make the classification problem "maximally hard" in the sense that weighted empirical risk of h_t, with respect to the updated weights $\mathcal{D}^{t+1}$, is the worse possible, namely $1/2$. Finally, the prediction rules vote in the committee with weights w_t.

Claim 5.4.1 For the weighting factor of $w_t = \frac{1}{2} ln\left(\varepsilon_t^{-1} - 1\right)$ and $\varepsilon = \sum_{i=1}^m \mathcal{D}_i^t \cdot \mathbb{1}\left[y_i \neq h_t\left(\mathbf{x}_i\right)\right]$ the weighted empirical risk of h_t with respect to $\mathcal{D}^{t+1}$ is $1/2$:

$$\sum_{i=1}^m \mathcal{D}_i^{t+1} \cdot \mathbb{1}\left[y_i \neq h_t\left(\mathbf{x}_i\right)\right] = \frac{1}{2}$$

Proof. Directly expressing the weighted empirical risk:

$$\begin{aligned}
\sum_{i=1}^m \mathcal{D}_i^{t+1} \cdot \mathbb{1}\left[y_i \neq h_i\left(\mathbf{x}_i\right)\right] &= \frac{\sum_{i=1}^m \mathcal{D}_i^t \exp(-\mathbf{w}_t y_i h_t(\mathbf{x}_i) \mathbb{1}\left[y_i \neq h_t(\mathbf{x}_i)\right])}{\sum_{j=1}^m \mathcal{D}_j^t \exp\left(-\mathbf{w}_t y_j h_t\left(\mathbf{x}_j\right)\right)} \\
&= \frac{\exp(\mathbf{w}_t \varepsilon_t)}{\exp(\mathbf{w}_t \varepsilon_t) + \exp(-\mathbf{w}_t)(1-\varepsilon_t)} \\
&= \frac{\varepsilon_t}{\varepsilon_t + \exp(-2\mathbf{w}_t)(1-\varepsilon_t)} \\
&= \frac{\varepsilon_t}{\varepsilon_t + \frac{\varepsilon_t}{1-\varepsilon_t}(1-\varepsilon_t)} = \frac{1}{2}
\end{aligned}$$

■

5.4.2 PAC View of Boosting - Weak Learnability

Historically, Boosting appeared as an answer to a fascinating question for which we first need to define **Weak Learnability**:

Definition 5.4.1 — γ-weak-learner. A learning algorithm $\mathcal{A}$ is a γ-weak-learner for an hypothesis class $\mathcal{H}$ if there exists a function $m_{\mathcal{H}} : (0,1) \to \mathbb{N}$ such that for every $\delta \in (0,1)$, for every distribution $\mathcal{D}$ over the sample space $\mathcal{X}$, and for every labeling function $f : \mathcal{X} \to \{\pm\}$, if the realizabiity assumption holds with respect to $\mathcal{H}, \mathcal{D}, f$, then when running $\mathcal{A}$ on a training sample of $m \geq m_{\mathcal{H}}(0,1)$ i.i.d samples drawn according to $\mathcal{D}$ and labeled by f, the algorithm returns an hypothesis $h_S = \mathcal{A}(S)$ such that with probability at least $1-\delta$ (with respect to choice of the training sample S), we have $L_{\mathcal{D},f}(h_S) \leq 1/2 - \gamma$.

Definition 5.4.2 An hypothesis class $\mathcal{H}$ is γ-weak-learnable if there exits a γ-weak-learner for $\mathcal{H}$.

How is this different than PAC-learnability? If an hypothesis class $\mathcal{H}$ is PAC-learnable, then for **every** (ε, δ) there exists a learner $\mathcal{A}$. This means that we can learn and generalize a labeling function from $\mathcal{H}$ to any accuracy ε we want. But if $\mathcal{H}$ is γ-weak-learnable, for any δ **and just for** $\varepsilon = 1/2 - \gamma$ there is a learner $\mathcal{A}$. We may not be able to find a learner that has better accuracy (lower ε).

The question that motivated Boosting was the following:

- Suppose that $\mathcal{H}$ is PAC-learnable. Then we know that the rule $ERM_{\mathcal{H}}$ will learn (namely, will be probably approximately correct etc) with a near-minimal number of samples.
- But what if $ERM_{\mathcal{H}}$ is computationally hard? (we've seen examples)
- Assume we can find a **simple** hypothesis class (a "base hypothesis class") $\mathcal{H}_{base}$, such that $ERM_{\mathcal{H}_{base}}$ (choosing the hypothesis in $\mathcal{H}_{base}$ with lowest empirical risk) is computationally efficient, and is γ-weak-learner for $\mathcal{H}$ for some γ.
- This means that we have a computationally efficient way to learn with accuracy $1/2 - \gamma$, for some γ. Maybe we can't find an efficient learner with better γ.
- Is there a way to **boost** $ERM_{\mathcal{H}_{base}}$ in a computationally efficient way, and create a computationally efficient learner $\mathcal{A}$ which is close to minimizing ERM over $\mathcal{H}$?

For example, think about Decision trees. We saw that the ERM learner is not computationally feasible on this hypothesis class. But a small tree may be able to achieve accuracy (over a sample labeled by a larger tree) which is not great, but better than random. Well, as the following theorem shows, Adaboost does just that (for the full proof refer to UML 10.2)

Theorem 5.4.2 Let S be a training set. Assume that at each iteration of Adaboost, the base learner returns a prediction rule (hypothesis h_t) for which the weighted empirical risk satisfies:

$$\sum_{i=1}^{m} \mathcal{D}_i^t \mathbb{1}\left[y_i \neq h(\mathbf{x}_i)\right] \leq \frac{1}{2} - \gamma$$

Then the (standard, non-weighted) empirical risk of the output prediction rule of Adaboost, h_{boost}, (the weighted committee vote) satisfies:

$$L_S(h_{boost}) \equiv \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\left[y_i \neq h_{boost}(\mathbf{x}_i)\right] \leq e^{-2\gamma^2 T}$$

5.4.3 Bias-Variance in Boosting

The hope is, of course, that we're not overfitting, so that low empirical risk will imply low generalization loss. Suppose we run T iterations of Adaboost over a learner $\mathcal{A}_{base}$ that returns hypothesis from $\mathcal{H}_{base}$. What is the effective hypothesis class we have now, and how large is it? Well, Adaboost with T iterations will return a function from the hypothesis class

$$\mathcal{H}_T = \left\{ \mathbf{x} \mapsto \sum_{t=1}^{T} w_t h_t(\mathbf{x}) \,|\, w_1 \dots w_T \in [0,\infty),\ \sum_t w_t = 1,\ h_1 \dots, h_T \in \mathcal{H}_{base} \right\}$$

namely convex combinations of hypotheses from $\mathcal{H}_{base}$. So $\mathcal{H}_t$ becomes larger (contains more functions) as T grows. But it doesn't grow too fast with T.

For example, we have a canonical way to measure the "size" of $\mathcal{H}_T$. While we won't go into the details, under certain conditions, $VCdim(\mathcal{H}_T)$ is roughly $T \cdot VCdim(\mathcal{H}_{base})$. So we can expect Boosting to increase the variance (compared with the base learner) as T increases, but "not too fast".

On the other hand, it's clear that Boosting decreases bias - that is obvious from the fact that the empirical risk decreases as T grows - indeed $\mathcal{H}_T$ is able to come closer and closer to the labeling function on the training set. And the fact that empirical risk decreases **exponentially** with T tells us that bias decreases quite quickly. Overall, Boosting typically decreases bias much faster than it increases variance, which is why it typically improves generalization loss quite dramatically. But a question remains: If we use T too large, will boosting overfit?

6. Regularization, Model Selection and Model

6.1 Regularization

Throughout the course we have discusses multiple hypothesis classes and learners for choosing an hypothesis $h_s \in \mathcal{H}$ from a given hypothesis class $\mathcal{H}$. In most of these cases choosing h_S was based on minimizing some cost function $\mathcal{F}_S(h)$ over $h \in \mathcal{H}$. This means that $h_S := \mathcal{A}_0(S)$ is given by:

$$h_S := \underset{h \in \mathcal{H}}{argmin} \mathcal{F}_S(h)$$

The function $\mathcal{F}$ measures how well h fits the training sample S. We can call $\mathcal{F}_S(h)$ the **fidelity term**. We have seen a few examples for such learners already:

- In linear regression, $\mathcal{F}_S(h)$ measures the **RSS**.
- In logistic regression, $-\mathcal{F}_S(h)$ is the **likelihood** of the model (which we want to maximize).
- In SVM, $-\mathcal{F}_S(h)$ is the **margin** of the hyperplane (which we want to maximize).

With the simplest example for learners that minimize a cost function are of course the **ERM** learners. In any ERM based learner, we define some loss function $\ell(\cdot,\cdot)$ and define the empirical risk induced by ℓ, with respect to the training sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ to be:

$$L_S(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i)$$

For all of these the fidelity term $\mathcal{F}_S(h)$ is the empirical risl $L_S(h)$.

For the different hypothesis classes we have also discussed how their richness and expressiveness governs the bias-variance tradeoff. If the hypothesis class $\mathcal{H}$ is "too large", we are concerned that minimizing $\mathcal{F}$ over $\mathcal{H}$ may lead to over-fitting. Namely, we are concerned our learner will output

an hypothesis h_S which will not generalize well, as it is too well adapted to the particular training sample S.

One way to solve this problem would be to restrict $\mathcal{H}$. However, in such a case we are deliberately harming our performance. Instead, we would like to keep $\mathcal{H}$ large, but find a way to tell our learner $\mathcal{A}_0$ "Prefer simpler hypotheses but if you find a complex one that is *really* worth it - take it". To achieve this, we change the optimization problem that $\mathcal{A}_0$ uses to choose h_S. We introduce an additional term to the problem. We choose $\lambda \geq 0$ and define the learner $\mathcal{A}_\lambda : S \mapsto \mathcal{H}$ by:

$$h_S := \underset{h \in \mathcal{H}}{argmin}\, \mathcal{F}_S(h) + \lambda \mathcal{R}(h)$$

The term $\mathcal{R}$ is called the **regularization term**. If choosing $\mathcal{R}$ wisely then $\mathcal{R}(h)$ will measure the "complexity" of the hypothesis h. The more complicated the hypothesis h, the larger $\mathcal{R}(h)$ will be. So we see that in minimizing $\mathcal{F}_S(h) + \lambda \mathcal{R}(h)$ we now have a **trade-off**:

- On one hand, more complicated h, the better it can describe the training sample S, so the fidelity term $\mathcal{F}_S(h)$ will be smaller.
- On the other hand, the more complicated h, the larger $\mathcal{R}(h)$ will be.

And conversely, the simpler h, the larger the fidelity term $\mathcal{F}_S(h)$ (as it won't be able to describe the training sample very well) but the smaller $\mathcal{R}(h)$ will be. So λ is a parameter the coverns the trade-off:

- For $\lambda = 0$, we have no regularization and are back to finding a minimizer of $\mathcal{F}_S(h)$.
- For $\lambda \to \infty$, the minimization problem pays no attention to the fidelity term and just wants to find the simplest possible $h \in \mathcal{H}$.
- Any finite value $\lambda \in (0, \infty)$ defines a specific trade-off between the need for fidelity (small $\mathcal{F}_S(h)$) and the need for simplicity of h (small $\mathcal{R}(h)$).

So, when we add regularization to the learner $\mathcal{A}_0 : S \mapsto \text{argmin}_{h \in \mathcal{H}} \mathcal{F}_S(h)$, we won't find a minimizer of $\mathcal{F}_S(h)$ over $\mathcal{H}$. We are hoping that a minimizer of the regularized objective function $\mathcal{F}_S(h) + \lambda \mathcal{R}(h)$ (which does not minimize $\mathcal{F}_S(h)$) will generalize better. This is because, the larger the value of λ, the simpler this minimizer will be.

We therefore get a **family** of learners $\{\mathcal{A}_\lambda\}_{\lambda \in [0,\infty)}$. The regularization parameter λ controls the bias-variance tradeoff: for $\lambda = 0$ we get the most variance and least bias (most complicated h); for $\lambda \to \infty$ we get the least variance and most bias (simplest h).

R We already saw one example for regularization - Soft SVM. Recall that the Soft-SVM classifier classifies using the half-space $\mathbf{w}^\perp$ where $\mathbf{w}$ is a solution to the optimization problem:

$$\begin{aligned} \text{minimize} \quad & \lambda ||\mathbf{w}||^2 + \frac{1}{m} \sum_i \xi_i \\ \text{subject to} \quad & y_i \cdot (\langle \mathbf{x}_i, \mathbf{w} \rangle + b) \geq 1 - \xi_i \text{ and } \xi_i \geq 0 \end{aligned}$$

Here, the fidelity term is $||\mathbf{w}||^2$. The smaller $||\mathbf{w}||^2$, the better we fit the training data (in the sense of larger margin). As the total margin is proportional to $1/||\mathbf{w}||$, so that minimizing $||\mathbf{w}||^2$ means maximizing the margin. The regularization term is $\frac{1}{m}\sum_i \xi_i$. The smaller this term is, the "simpler" the hypothesis since we allow less violations of the margin.
Note that here λ is placed on the fidelity term and not on the regularization term.

Regression Trees - Add definitions to Decision Trees under classification. Add adaptation of algorithm as exercise Add pruning of trees - Maybe add it under classification. Just refer to here for ideas (and teach when getting here)

6.1.1 Subset Selection

Let us revisit linear regression and the OLS estimator. We defined the hypothesis class of linear regression 2.1:

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

and given a regression problem $\mathbf{X}, \mathbf{y}$, we select $h_S \in \mathcal{H}_{reg}$ by using the OLS estimator (which we have derived both as ERM for the square loss and using Maximum Likelihood assuming Gaussian noise). When doing so we assumed that the training sample size m aws not smaller than the number of features d, and even preferred that $m \gg d$, since of $m \sim d$ the variance of the OLS estimator can be large.

However, in modern learning problems based on d features, very often we are in a situation where d can be quite large. Linear regression was invented when features were measured and recorded manually. In the last few decades it became very easy to collect features and record them automatically, and a typical regression problem can easily have $d \sim m$ or even $d \gg m$.

When $d \sim m$ or, worse, $d \gg m$, we will have correlated features. The coefficients fitted by linear regression will be poorly determined. For example, a large positive coefficient for some feature can be canceled out by a large negative coefficient for an almost-parallel feature. So the linear regression learner we saw should not be used for $m \sim d$ and cannot be used for $d > m$.

Therefore, we would like to devise a method where we keep only a subset of the features. Let $k \leq d$ be the desired number of features, then we could solve the following optimization problem:

$$\begin{array}{ll} \text{minimize} & ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2 \\ \text{subject to} & \sum_{i=1}^{d} \mathbb{1}[\mathbf{w}_i \neq 0] = k \end{array} \tag{6.1}$$

That is, find a coefficients vector $\mathbf{w}$ that minimizes the *RSS* under the restriction of using exactly k features. This is exactly what the Best-Subset Selection algorithm does (reference to pseudo). It iterates all possible combinations of k features, fits an OLS model for each, and returns the model (i.e. a vector $\mathbf{w}$) achieving the lowers loss. A related, but not equivalent, problem introduces a regularization term over the number of used features:

$$\underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{argmin} \ ||w_0 \mathbf{1} + \mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \lambda \, ||\mathbf{w}||_0 \tag{6.2}$$

where $||\mathbf{w}||_0 := \sum_i \mathbb{1}[\mathbf{w}_i \neq 0]$. Notice how in both optimization problems we do not include the intercept (w_0) in the restriction on the number of features as the intercept is not one of the features per se.

Add pseudo of Best-subset-selection

As k, the number of features in the model, gives some measure of the complexity of the hypothesis, we can think of the family of learners $\left\{ \mathcal{A}_k^{best-subset} | 0 \leq k \leq d \right\}$. As we change k we move on the

bias-variance trade-off. Unfortunately, even for a single value of k, we must go over all $\binom{d}{k}$ subsets of k features. This is an NP-Hard problem and as such cannot be solved efficiently.
Add Forward/Backaward Selection algorithms?

6.1.2 Ridge (ℓ_2) Regularization

In addition to the computational challenges imposed by the best-subset regularization, it also often suffers from high variance as features are included or excluded based on the single specific training-set we have. To cope with both problems, we often use different *shrinkage methods* where we restrict the values of the coefficients, shrinking them towards zero. One such method is the Ridge regression, which imposes a $||\cdot||_2$ penalty on the coefficients:

$$\widehat{\mathbf{w}}_\lambda^{ridge} := \underset{w_0\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d}{argmin} ||w_0\mathbf{1}+\mathbf{X}\mathbf{w}-\mathbf{y}||^2+\lambda\,||\mathbf{w}||_2^2 \quad \lambda\geq 0 \tag{6.3}$$

So $\lambda\geq 0$ is the complexity parameter that controls the amount of shrinkage. For $\lambda=0$ we get the OLS solution (the standard linear regression solution). As $\lambda\to\infty$ we penalize more and more on the size of the coefficients vector, driving the solution $\widehat{\mathbf{w}}_\lambda^{ridge}\to 0$. As λ increases the bias increases (we restrict ourselves to specific sets of solutions) and variance decreases (solution is not based solely on training-set).

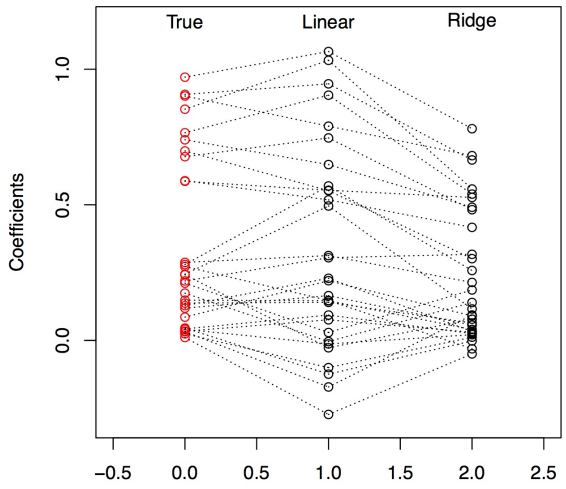


Figure 6.1: *Add caption*

better removal of penalty over intercept - ESL - page 64 Though the Ridge optimization problem is a convex problem, and specifically a case of quadratic programming, it has in-fact a neat closed form.

Theorem 6.1.1 Let $\mathbf{X},\mathbf{y}$ be a regression problem and let $\lambda\geq 0$. The solution for the Ridge Regression problem 6.3 is given by:

$$\widehat{\mathbf{w}}_\lambda^{ridge} := \left(\mathbf{X}^\top\mathbf{X}+\lambda I_d\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

Proof. For $\lambda=0$ then indeed $\widehat{\mathbf{w}}_\lambda^{ridge}=\widehat{\mathbf{w}}^{ols}$. Let $\lambda>0$ then to find a minimizer of the objective

function, we begin with equating it's derivative in each coordinate to zero:

$$\frac{\partial \left|\left|w_0\mathbf{1}+\mathbf{X}\mathbf{w}-\mathbf{y}\right|\right|^2+\lambda\left|\left|\mathbf{w}\right|\right|_2^2}{\partial \mathbf{w}_k}=-2\sum_{i=1}^{m}\left(y_i-\sum_{j=1}^{d}\mathbf{x}_{ij}\mathbf{w}_j\right)\mathbf{x}_k+2\lambda\mathbf{w}_k=0 \quad i=1,\ldots,d$$

which in matrix notation is:

$$\begin{gathered}\left[\mathbf{X}^\top\mathbf{y}\right]_k-\left[\mathbf{X}^\top\mathbf{X}\mathbf{w}\right]_k-\left[\lambda\mathbf{w}\right]_k=0\\ \Downarrow\\ \mathbf{X}^\top\mathbf{y}=\mathbf{X}^\top\mathbf{X}\mathbf{w}+\lambda I_d\mathbf{w}\\ \Downarrow\\ \mathbf{X}^\top\mathbf{y}=\left(\mathbf{X}^\top\mathbf{X}+\lambda I\right)\mathbf{w}\end{gathered}$$

■

As $\mathbf{X}^\top\mathbf{X}+\lambda I$ is non-singular (even if $\mathbf{X}^\top\mathbf{X}$ is not of full rank) it has an inverse and hence the solution is given by:

$$\widehat{\mathbf{w}}_\lambda^{ridge}=\left(\mathbf{X}^\top\mathbf{X}+\lambda I\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

Ridge regression could also be solved using the SVD method for OLS weights, which can be generalized.

Claim 6.1.2 Let $\mathbf{X},\mathbf{y}$ be a regression problem and $\mathbf{X}=U\Sigma V^\top$ be the SVD of $\mathbf{X}$. The Ridge estimator is given by:

$$\widehat{\mathbf{w}}_\lambda^{ridge}=U\Sigma_\lambda V^\top\mathbf{y},\quad [\Sigma_\lambda]_{ii}=\frac{\sigma_i}{\sigma_i^2+\lambda}$$

Proof. Let us replace $\mathbf{X}$ with its SVD:

$$\begin{array}{rclrcl}\widehat{\mathbf{w}}_\lambda^{ridge} &=& \left(\mathbf{X}^\top\mathbf{X}+\lambda I\right)^{-1}\mathbf{X}^\top\mathbf{y} &=& \left(\left(U\Sigma V^\top\right)^\top U\Sigma V^\top+\lambda I\right)^{-1}\left(U\Sigma V^\top\right)^\top\mathbf{y}\\ &=& \left(V\Sigma U^\top U\Sigma V^\top+\lambda I\right)^{-1}V\Sigma U^\top\mathbf{y} &=& \left(V\Sigma^2V^\top+\lambda VV^\top\right)^{-1}V\Sigma U^\top\mathbf{y}\\ &=& V\left(\Sigma^2+\lambda I\right)^{-1}V^\top V\Sigma U^\top\mathbf{y} &=& V\Sigma_\lambda U^\top\mathbf{y}\end{array}$$

where we denote $[\Sigma_\lambda]_{ii}=\frac{\sigma_i}{\sigma_i^2+\lambda}$ and σ_i are the singular values of $\mathbf{X}$. ■

Explain that is a biased estimator but reduces the variance. Maybe take question from homework and put here instead. Might be beneficial to show this in recitation instead
Add regularization path explanation

6.1.3 Lasso (ℓ_1) Regularization

By introducing the ridge regularizer we were able to apply linear regression when $d>m$ or when $\mathbf{X}^\top\mathbf{X}$. In addition, we saw that even though the ridge estimator is a biased estimator it achieves a lower variance compared to the OLS estimator. However, as evident from the regularization path, the ridge estimator does not do what best-subset does. That is, it cannot **select** a specific subset of features to use for regression.

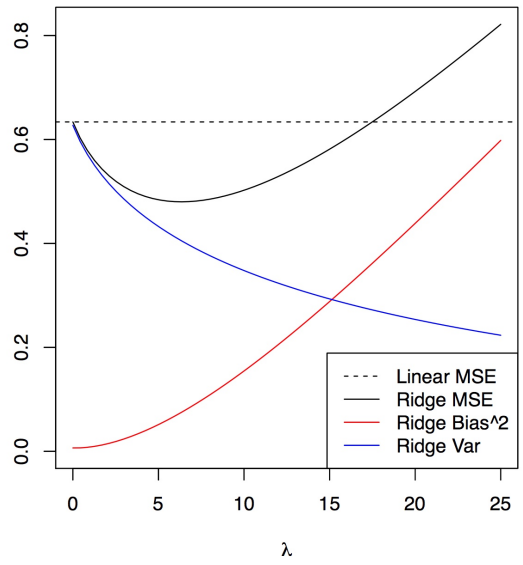


Figure 6.2: *Add caption*

The *Least Absolute Shrinkage and Selection Operaotr* (Lasso) attempts to achieve exactly that. This regularization method uses the ℓ_1 norm rather than the ℓ_2 norm:

$$\widehat{\mathbf{w}}_{\lambda}^{lasso} := \underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{argmin} \; ||w_0 \mathbf{1} + \mathbf{X}\mathbf{w} - \mathbf{y}||^2 + \lambda \, ||\mathbf{w}||_1 \quad \lambda \geq 0 \tag{6.4}$$

This optimization problem is still convex and therefore can be solved efficiently. Similar to Ridge setting $\lambda = 0$ we get the OLS solution and setting $\lambda \to \infty$ will shrink the coefficients $\widehat{\mathbf{w}}_{\lambda}^{lasso} \to 0$. However the manner of shrinkage is very different between the two problems. The lasso solution, $\widehat{\mathbf{w}}_{\lambda}^{lasso}$, is **sparse**. That is, it has zero entries in the vector. Therefore, the larger λ, typically the more zeros $\widehat{\mathbf{w}}_{\lambda}^{lasso}$will have, effectively defining a subset of features that are used by the model (a feature corresponding a coefficient of zero is not used by the model). We often refer to this subset of features as the "active set". As we are considering problems where d can be very large, it is hard to interpret the model. Thus, the Lasso has an important advantage in interpretability over Ridge, as it selects for us a subset of variables.

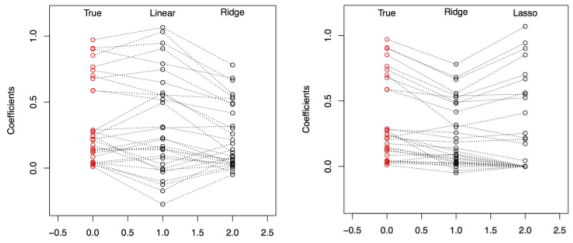


Figure 6.3: *Add caption*

Add comparison of regularization path between ridge and lasso

6.1.3.1 Convexity vs. Sparsity

We have discussed three regularization methods for regression problems using the terms: $||\cdot||_0, ||\cdot||_1, ||\cdot||_2$ (which we will also denote by ℓ_0, ℓ_1, ℓ_2). We saw that for both ℓ_1, ℓ_2 we still have a convex optimization problem whereas for the ℓ_0 term the problem is non-convex. In addition, we stated that unlike ℓ_2, when using ℓ_1 we typically introduce sparsity to the solution. Let us expand this discussion for the L_q familty of norms:

$$\text{For } 0 < q \in \mathbb{R}, x \in \mathbb{R}^d \quad ||x||_q := \left(\sum_{i=1}^{d} (x_i)^q \right)^{\frac{1}{q}} \tag{6.5}$$

If applying any norm in this family as the regression regularization term then:

- Sparsity - For any $q \leq 1$ we typically obtain sparse solutions and therefore our active set of features is smaller than d.
- Convexity - For any $q \geq 1$ the optimization problem (when we use the RSS loss as the fidelity term) is convex. Therefore the problem can be solved efficiently.

The *Lasso* regression uses $q = 1$ and as such achieves **both** sparsity and convexity. But why are $L_{q\leq 1}$ norms obtain their sparsity? This is in fact based on the shapes of their unit balls.

Definition 6.1.1 For a norm $||\cdot||$ on $\mathbb{R}^d$, a ball of radius ρ (around the origin) is the set: $B_\rho := \{\mathbf{w} \in \mathbb{R}^d | \, ||\mathbf{w}|| \leq \rho\}$. The unit ball of a norm is the ball of radious $\rho = 1$.

For $L_{q>1}$, such as in the case of the Euclidean norm, the unit ball has no corners. In contrast, the unit balls of $L_{q\leq 1}$ have corners. Specifically, the unit ball of the ℓ_1 norm over $\mathbb{R}^d$ has $2d$ corners.

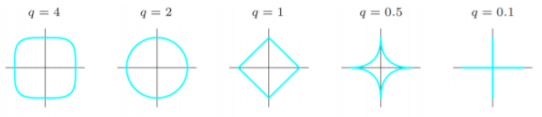


FIGURE 3.12. *Contours of constant value of* $\sum_j |\beta_j|^q$ *for given values of* q.

Figure 6.4: *Add caption*

Now, let us revisit the Ridge and Lasso optimization problems. Instead of the unconstrained forms seen in 6.4 and 6.3 consider the equivalent constraint versions:

minimize $w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d$	$\|\|w_0\mathbf{1} + \mathbf{X}\mathbf{w} - y\|\|^2$	minimize $\mathbf{w}_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d$	$\|\|w_0\mathbf{1} + \mathbf{X}\mathbf{w} - y\|\|^2$
subject to	$\|\|\mathbf{w}\|\|_2^2 \leq \rho$	subject to	$\|\|\mathbf{w}\|\|_1 \leq \rho$

Now, let us fix some $\rho \in \mathbb{R}$ and ask where will we find the $\widehat{\mathbf{w}}_\rho^{ridge}$ and $\widehat{\mathbf{w}}_t^{lasso}$ solutions. As the fidelity term is a quadratic form in $\mathbf{w}$, its level sets are ellipsoids. Suppose ρ is very large, so there is effectively no constrain, the solution of the minimization problem will be the OLS solution. As we restrict the solution more and more, by making ρ smaller and smaller, we are limiting the solution to be inside the norm ball. By definition, the solution will be found where one of the level sets of the fidelity term intersects with the ball of radius ρ. When we consider the $L_{q\leq 1}$ norms (such as the ℓ_1 in Lasso) this intersection typically happens at one of the corners of the norm ball. As these corners take place on the axes they correspond to **sparse** solutions.

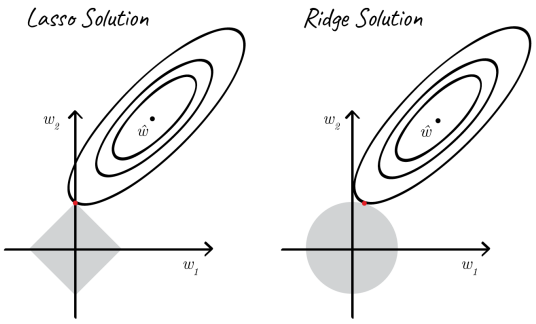


Figure 6.5: *Add caption*

6.1.4 The Orthogonal Design Case

Here is another way to understand why Lasso solutions are sparse. Consider the case where all features are orthogonal to each other, namely $\mathbf{X}^\top\mathbf{X} = I_d$. In some sense, this is the simplest setup for regression problems, where we can write $\mathbf{y}$ as a linear combination of **orthonormal** vectors. This is also known as an orthogonal design. In such setup, we have a closed form solution for $\widehat{\mathbf{w}}_\lambda^{subset}, \widehat{\mathbf{w}}_\lambda^{ridge}, \widehat{\mathbf{w}}_\lambda^{lasso}$ based on the $\widehat{\mathbf{w}}^{ols}$ solution.

Let us define two *thresholding* functions.

Definition 6.1.2 Let the hard- and soft-threshold (at λ) be the functions $\eta_\lambda^{hard}, \eta_\lambda^{soft} : \mathbb{R} \to \mathbb{R}$ defined by:

$$\eta_\lambda^{hard} := \mathbb{1}\left[|x| - \lambda\right] \cdot x \qquad \eta_\lambda^{soft} := sign\left(x\right)\left[|x| - \lambda\right]_+ = \begin{cases} x - \lambda & x \geq \lambda \\ 0 & |x| < \lambda \\ x + \lambda & x \leq \lambda \end{cases}$$

These functions define a manner of shrinking an input value x towards 0, depending on λ. The hard-thresholding zeros inputs in the range of $(-\lambda, +\lambda)$ and leaves inputs outside of this range unchanged. The shoft-thresholding shrinks inputs out of the $(-\lambda, +\lambda)$ range, and zeros inputs that are within the range.

Claim 6.1.3 Let X, y be an orthogonal design matrix and a response vector. Denote $\widehat{w}$ the OLS solution. The lasso regularization optimization problem takes the form of $\widehat{w}^{lasso}\left(\lambda\right) = \eta_\lambda^{soft}\left(\widehat{w}\right)$

Proof. Recall that the OLS solution is $\hat{\mathbf{w}} = (\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}\mathbf{y} = \mathbf{X}\mathbf{y}$. Write the objective function:

$$\begin{aligned} f_{\ell_1}\left(\mathbf{w}\right) = \tfrac{1}{2}\left|\left|\mathbf{y} - \mathbf{X}\mathbf{w}\right|\right|_2^2 + \lambda\left|\left|\mathbf{w}\right|\right|_1 &= \tfrac{1}{2}\left(\left|\left|\mathbf{y}\right|\right|^2 - 2\mathbf{y}^\top\mathbf{X}\mathbf{w} + \mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w}\right) + \lambda\left|\left|\mathbf{w}\right|\right|_1 \\ &= \tfrac{1}{2}\left(\left|\left|\mathbf{y}\right|\right|^2 + \left(\mathbf{w}^\top - 2\widehat{\mathbf{w}}^\top\right)\mathbf{w}\right) + \lambda\left|\left|\mathbf{w}\right|\right|_1 \\ &= \tfrac{1}{2}\left|\left|\mathbf{y}\right|\right|^2 + \textstyle\sum_{j=1}^d\left(\frac{1}{2}\mathbf{w}_j^2 - \widehat{\mathbf{w}}_j\mathbf{w}_j + \lambda\left|\mathbf{w}_j\right|\right) \\ &= \tfrac{1}{2}\left|\left|\mathbf{y}\right|\right|^2 + \textstyle\sum_{j=1}^d\left(\frac{1}{2}\mathbf{w}_j - \widehat{\mathbf{w}}_j + \lambda sign\left(\mathbf{w}_j\right)\right)\mathbf{w}_j \end{aligned}$$

Let us minimize the expression for each $\mathbf{w}_j$ taking into account the value of λ. So:

$$\frac{\partial \frac{1}{2}||\mathbf{y}-\mathbf{X}\mathbf{w}||+\lambda||\mathbf{w}||_1}{\partial \mathbf{w}_j} = \frac{\partial\left(\frac{1}{2}\mathbf{w}_j-\widehat{\mathbf{w}}_j+\lambda sign\left(\mathbf{w}_j\right)\right)\mathbf{w}_j}{\partial \mathbf{w}_j} = \mathbf{w}_j - \widehat{\mathbf{w}}_j + \lambda sign\left(\mathbf{w}_j\right) = 0$$
$$\Downarrow$$
$$\mathbf{w}_j = \widehat{\mathbf{w}}_j - \lambda sign\left(\mathbf{w}_j\right)$$

Let us divide into cases:

- If $\left|\widehat{\mathbf{w}}_j\right| < \lambda$ then $\mathbf{w}_j = 0$. That it because:
  - If $\mathbf{w}_j < 0$ then $0 > \mathbf{w}_j = \widehat{\mathbf{w}}_j + \lambda$, which $\forall \left|\widehat{\mathbf{w}}_j\right| < \lambda$ the right-hand side is positive in contradiction.
  - If $\mathbf{w}_j > 0$ then $0 < \mathbf{w}_j = \widehat{\mathbf{w}}_j - \lambda$, which $\forall \left|\widehat{\mathbf{w}}_j\right| < \lambda$ the right-hand side is negative in contradiction.
- If $\widehat{\mathbf{w}}_j \geq \lambda$ then $\mathbf{w}_j = \widehat{\mathbf{w}}_j - \lambda sign\left(\mathbf{w}_j\right)$ which is valid only for $\mathbf{w}_j \geq 0$ which means that $\mathbf{w}_j = \widehat{\mathbf{w}}_j - \lambda$.
- If $\widehat{\mathbf{w}}_j \leq -\lambda$ then $\mathbf{w}_j = \widehat{\mathbf{w}}_j - \lambda sign\left(\mathbf{w}_j\right)$ which is valid only for $\mathbf{w}_j \leq 0$ which means that $\mathbf{w}_j = \widehat{\mathbf{w}}_j + \lambda$.

Putting it all together we got that:

$$\mathbf{w}_j\left(\widehat{\mathbf{w}}_j,\lambda\right) = \begin{cases} \widehat{\mathbf{w}} - \lambda & \widehat{\mathbf{w}} \geq \lambda \\ 0 & |\widehat{\mathbf{w}}| < \lambda \\ \widehat{\mathbf{w}} + \lambda & \widehat{\mathbf{w}} \leq \lambda \end{cases} \quad \Longrightarrow \quad \widehat{\mathbf{w}}_j^{lasso}\left(\lambda\right) = \eta_\lambda^{soft}\left(\widehat{\mathbf{w}}_j^{ols}\right)$$

■

Similarly, we can obtain the Best-Subset and Ridge solutions:

$$\begin{aligned} \widehat{\mathbf{w}}_\lambda^{subset} &:= \eta_{\sqrt{\lambda}}^{hard}\left(\widehat{\mathbf{w}}^{ols}\right) \\ \widehat{\mathbf{w}}_\lambda^{ridge} &:= \widehat{\mathbf{w}}^{ols}/\left(1+\lambda\right) \end{aligned}$$

Namely, in the orthogonal design setup we can obtain the different solutions by applying a **univariate shrinkage function** to each coordinate of $\widehat{\mathbf{w}}^{ols}$ separately. Observe that in this case:

- The Best-Subset sets some coefficients to zero and leaves the rest untouched.
- The Lasso solution sets some coefficients to zero and shrinks the rest by λ.
- The Ridge solution simply multiplies by a scalar.

Therefore, for the Best-Subset and Lasso solutions, the solutions' sparsity grows as λ grows.

6.1.5 Regularized Logistic Regression

We have seen a few different (linear) regression methods using regularization. We can also apply these regularization terms to other regression models such as the logistic regression. Similarly to linear regression, if $m \sim d$ or $d > m$, also for logistic regression optimization will be numerically unstable, might have parallel feature vectors with large coefficients of opposite signs and hard for interpretation. All these problems are alleviated by adding a regularization term.

Recall that the logistic regression classifier find the coefficients vector by solving:

$$\widehat{\mathbf{w}} := \underset{w_0\in\mathbb{R},\mathbf{w}\in\mathbb{R}^d}{argmax} \sum_{i=1}^{m}\left[y_i\left(\left\langle \mathbf{x}_i,\mathbf{w}\right\rangle + w_0\right) - log\left(1+e^{w_0+\left\langle \mathbf{w},\mathbf{x}_i\right\rangle}\right)\right]$$

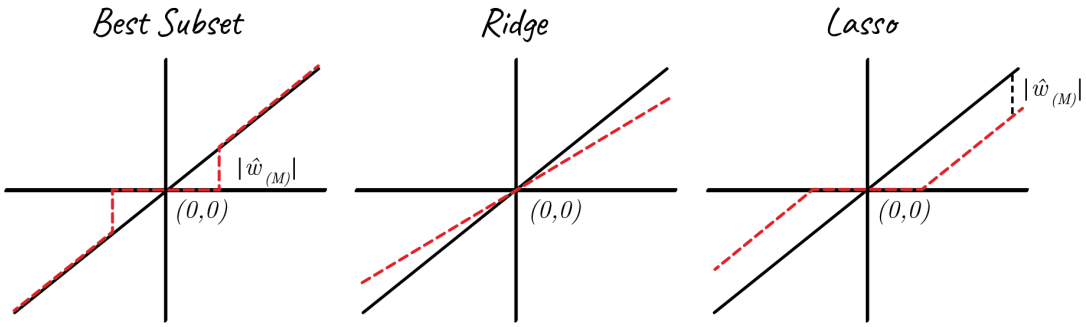


Figure 6.6: *Add caption*

We can add the ℓ_1 regularization term to the above fidelity term to obtain the ℓ_1-regularized logistic regression classifier:

$$\widehat{\mathbf{w}} := \underset{w_0 \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^d}{argmax} \sum_{i=1}^{m} \left[y_i \left(\langle \mathbf{x}_i, \mathbf{w} \rangle + w_0 \right) - log \left(1 + e^{w_0 + \langle \mathbf{w}, \mathbf{x}_i \rangle} \right) + \lambda \, ||\mathbf{w}||_1 \right]$$

This is still a convex optimizatoin problem, and fast specialized solvers are available. The ℓ_1-regularized logistic regression classifier is a very powerful classifier over $\mathcal{X} = \mathbb{R}^d$. It has low variance, we are able to control the bias-variance tradeoff (by choosing λ) and is it very interpretable.

6.2 Model Selection and -Evaluation

So far, we have discussed several learning algorithms and meta-algorithms that can be applied over the existing learning algorithms. In the coming section we will be answering the following questions:

- How to **select** a model? For different algorithms we have described the existance of a family of learners where we defined some "tuning" hyper-parameter. For example, k the number of neighbors used in the k-NN algorithm; the maximal tree depth and pruning regularization in CART; or the regularization lambda in soft-SVM and the different regularized linear- and logistic- regression problems.

  In the case of meta-algorithms such as bagging or boosting, in addition to the base learners' tuning parameters, we need to choose the number of bagging/bootstrapping iterations B. If we are using de-correlation in bagging, there might be additional tuning parameters (e.g the number k of allowed coordinated for a split in the Random Forest algorithm).

- How to **estimate** the performance of a chosen model? Before applying the chosen model on new samples we would like to estimate how will it perform on a new independent set of samples. If our estimation of the generalization error for the selected model is poor, perhaps we are working with the wrong learning algorithm for our problem, and should therefore select a different candidate. In addition, often we are interested in known how our chosen learner will perform before we begin using it.

The performance we would like to estimate, which would also influence the selection of the model, is the generalization error. Assume some loss function $\ell(\cdot,\cdot)$ then we defined the empirical risk of an hypothesis $h : \mathcal{X} \to \mathcal{Y}$ simple as the average loss over the training sample:

$$L_S(h) \coloneqq \frac{1}{m}\sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i) \quad S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$$

The generalization error (also reffered to as *test error*), that is the predicted error over an independent sample set depends on our data-generation model:

- No data-generation model: If we are not assuming any model to describe how the data is generated we simply define the generalization error as the average loss over the test set:

$$L(h) \coloneqq \sum_{i=1}^{|T|} \ell(h(\mathbf{x}_i), y_i)$$

- Probability distribution over $\mathcal{X}$ and unknown labeling function $f : \mathcal{X} \to \mathcal{Y}$: When assuming a model as the PAC model, the generalization error is the expected error over sampling from the distribution:

$$\mathbb{E}_{x\sim\mathcal{D}}[\ell(h(\mathbf{x}), f(\mathbf{x}))]$$

- Probability distribution over $\mathcal{X} \times \mathcal{Y}$: When assuming a model as the Agnostic PAC model, the generalization error is the expected error over sampling a sample-label pair from the distribution:

$$L_{\mathcal{D}}(h) \coloneqq \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(h(\mathbf{x}), y)]$$

Better conned Bias-Variance part - or not include it as we already covered it in the Bias-Variance chapter?

Once we understand what is the generalization error, the next step is to find a way to **correctly estimate** it. Suppose we follow the following procedure: Given a supervised batch learning setting, where our training sample is $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, let us perform model selection and -evaluation over a family of learning algorithms $\{\mathcal{A}_\alpha\}$ simply by using S:

- Training: Train each model over S to obtain $\{h_\alpha \coloneqq \mathcal{A}_\alpha(S)\}$
- Model selection: Choose the best model by using $\alpha^* \coloneqq argmin_\alpha L_S(h_\alpha)$
- Model evaluation: Estimate the generalization error of the chosen trained model h_{α^*} as $L_{\mathcal{D}}(h_{\alpha^*}) = L_S(h_{\alpha^*})$. Namely, the generalization error estimator is just the model's empirical risk over S.

This procedure will yield poor results. The selection is based on the empirical error of the model. As we have already seen in the bias-variance trade-off, the generalization error can be represented as the sum of both the bias- and the variance of the model. The model we select α^* will adapt as much as possible (under the restriction to some $\mathcal{H}$) to the training data. The more variance the learner has, the more it will be able to adapt to the particular properties of S. Out generalization error estimator will suffer from **optimism** (which is in fact the technical term for the difference between the empirical risk and generalization error). Thus, we need to devise a different method to estimate the generalization error. One that will suffer less from optimism.

6.2.1 Train-Validation-Test Scheme

The naive approach of using the finite training set S both for training the model and estimating it's future performance yields poor results. Suppose we had infinitely many samples, could we have done better? In such scenario we could use three different sets S (training), V (validation) and T (testing):

- Training: Train each model over S to obtain $\{h_\alpha := \mathcal{A}_\alpha(S)\}$
- Model selection: Choose the model minimizing the loss over the validation set: $\alpha^* := argmin_\alpha L_V(h_\alpha)$
- Model evaluation: Estimate the generalization error of the chosen trained model h_{α^*} as the loss over the test set. That is, $L_{\mathcal{D}}(h_{\alpha^*}) := L_T(h_{\alpha^*})$.

The introduction of a third set of samples, the validation set, which is used to select the final model decouples the training- from the model selection- stages. This provides us with an unbiased estimator of the generalization error which we can also use to bound the generalization error.

Claim 6.2.1 Let $h_S := \mathcal{A}(S)$ be the hypothesis returned by a learner over the training sample S, and let V be a new ("fresh") iid samples from $\mathcal{D}$. The empirical risk of h_S over V is an unbiased estimator of the generalization error of h_S.

Proof. Denote $m_V = |V|$. As the sample are identically distributed then:

$$\begin{aligned}
\mathop{\mathbb{E}}_{V\sim\mathcal{D}^{m_V}}[L_V(h_S)] &= \frac{1}{m_V}\sum_{i=1}^{m_V}\mathbb{E}_{V\sim\mathcal{D}^{m_V}}[\ell(h_S,(\mathbf{x}_i,y_i))] \\
&= \frac{1}{m_V}\sum_{i=1}^{m_V}\mathbb{E}_{(\mathbf{x}_i,y_i)\sim\mathcal{D}}[\ell(h_S,(\mathbf{x}_i,y_i))] \\
&= L_{\mathcal{D}}(h_S)
\end{aligned}$$

■

For the following, let us assume that the loss function is *bounded*. If the loss function is unbounded, the generalization error cannot be bounded. Suppose the loss function is bounded by 1.

Corollary 6.2.2 The generalization error of h_S is bound by:

$$\mathbb{P}\left[|L_V(h_S) - L_{\mathcal{D}}(h_S)| \le \sqrt{\frac{\log(2/\delta)}{2m_V}}\right] \ge 1-\delta$$

Proof. Recall from Hoeffding's inequality that if $X_1,\ldots,X_m$ are a set of bounded iid random variables such that $0 \le X_i \le 1$ and $\overline{X} = \frac{1}{m}\sum X_i$, then: $\mathbb{P}\left[\left|\overline{X} - \mathbb{E}\left[\overline{X}\right]\right| \ge \varepsilon\right] \le 2\exp\left(-2m\varepsilon^2\right)$. As V is a set of iid samples denote

$$Z_i := (\mathbf{x}_i, y_i)$$

the random variable of the selected pair. Since the samples are iid then $Z_1,\ldots,Z_{m_V}$ are iid too. Next, denote

$$X_i := \ell(h_S, Z_i)$$

the random variable of the loss over the Z_i sample. It holds that $X_1,\ldots,X_{m_V}$ are a set of iid random variables such that $0 \le X_i \le 1, \quad i = 1,\ldots,m_V$. Notice, that for $\overline{X} = \frac{1}{m_V}\sum X_i = L_V(h_S)$, as $L_V(h_S)$ is

an unbiased estimator of the generalization error, we directly get that:

$$\mathbb{P}\left[\left|L_V\left(h_S\right) - L_{\mathcal{D}}\left(h_S\right)\right| \geq \varepsilon\right] \leq 2\exp\left(2m_V\varepsilon^2\right)$$

By setting $\varepsilon = \sqrt{\frac{1}{2m_V}\ln\frac{2}{\delta}}$ we conclude that:

$$\mathbb{P}\left[\left|L_V\left(h_S\right) - L_{\mathcal{D}}\left(h_S\right)\right| \leq \sqrt{\frac{\log\left(2/\delta\right)}{2m_V}}\right] \geq 1-\delta$$

■

6.2.2 Cross Validation

In reality, splitting our finite set of samples into three sets is problematic. In the great majority of cases we are unwilling to decrease the size of our training set. We have already seen that training over a smaller set yields inferior results. Therefore, designating an additional portion of our limited number of samples just for validation isn't feasible. Instead we would like to come up with some method to use S for training but still do proper model selection and -evaluation.

Potentially the simplest method to do so is cross-validation (CV). Instead of thinking about S as a single set, let us think of it as a disjoint union of K equality sized sets, named folds. Then, for each $k = 1,\ldots,K$, we fit a model using all samples of S **except** samples belonging to the k'th fold. We then use the k'th fold to calculate the prediction error of the fitted model. Finally we report the estimated generalization error across all K folds. This method is called k-fold Cross Validation. The k'th fold, which is not used for training but only for evaluating the trained model functions as an unbiased estimator for the generalization error.

So this method can be used for selecting the tuning parameters. For each candidate $\mathcal{A}_\alpha$, we train it k times according to the CV procedure, each time leaving out one of the k folds. Then we choose α (and as such the learner $\mathcal{A}_\alpha$) whose average error (over the k validation sets) was lowest. Once we finished the model selection phase using CV, and have a fully specified learner $\mathcal{A}$, we train $\mathcal{A}$ **again** on the entire training sample S. This way we are still able to train $\mathcal{A}$ on as many points as possible.

The last step is using CV for model evaluation. Once we have a final, fully specified learner $\mathcal{A}$, we run k-fold CV and report the average error and standard deviation over the k folds. In this manner we supply both an estimation of the generalization error and a measure of accuracy for this estimate.

Choosing k, the number of folds

By using CV we have now introduced another "hyper-hyper"-parameter. How should we choose the value for k? If $k = 1$ then there is no CV. If $k = 2$ we split the data into two equally sized folds where we train only one half of the data. Therefore, the reported CV error will be larger than the true generalization error (also known out-of-sample error). If $k = m$, the sample size, it is called **leave-one-out** CV. Generally, if k is small we may be training on a dataset too small. As such the CV

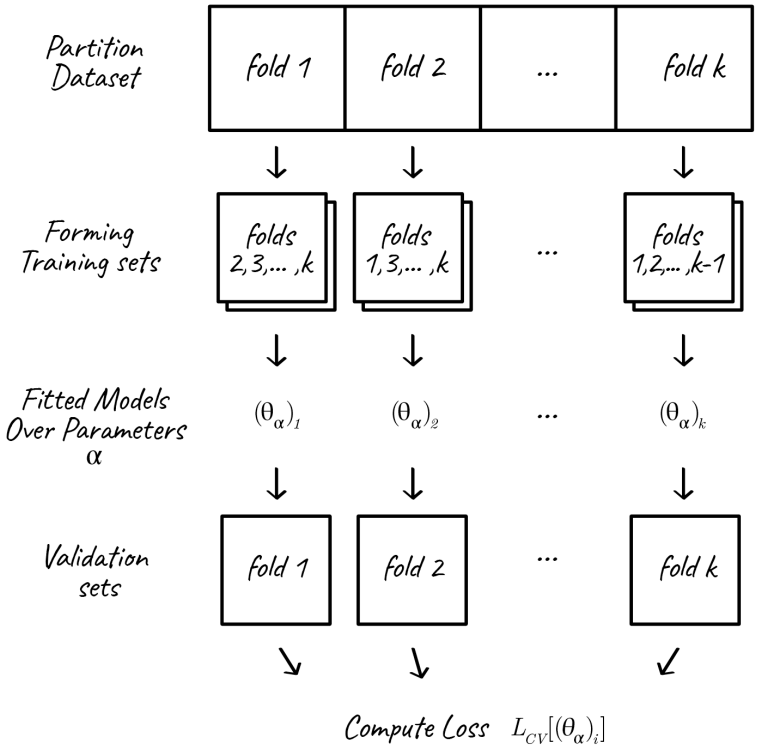


Figure 6.7: *Add caption*

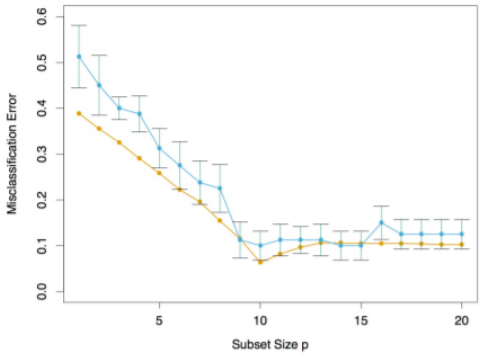


Figure 6.8: *Add caption*

error may be biased upwards. On the other hand, if k is large, the training samples are very similar to each other. As such the trained models are highly correlated, which might introduce high variances.

R Another consideration when using CV is computational. Since we train each model k times, the larger k, the more computations we perform.

6.2.3 Bootstrap

In subsection 5.3.1 we introduced the idea of bootstrapping, where by using re-sampling with replacement we can seemingly create new datasets. This method can also be used for estimating the generalization error using just the single training sample S. To do so let $\mathcal{A}_\alpha$ be some learner and $B \in \mathbb{N}$ the number of bootstrap samples to create. Very similar to the CV approach we could now:

- Draw a bootstrap sample $S^{(b)}$ by sampling m samples from S with replacement.
- By sampling $S^{(b)}$ we also obtain an independent test set $T^{(b)} := S \setminus S^{(b)}$ being the samples not chosen in the b-th bootstrap sample. These samples are also called the **out-of-bag** samples.
- Train $\mathcal{A}$ over $S^{(b)}$ to aquire an hypothesis h_S and test it over $T^{(b)}$.
- Finally, report the estimated mean and standard deviation of the generalization error, as measured over the B test sets.

6.2.4 Common Model Selection Mistakes

There are two very common mistakes when performing model evaluation using either of the methods seen before. The first causes over-estimation of the generalization error while the other causes under-estimation of the generalization error.

6.2.4.1 Over-estimating Generalization Error

Consider a family of learners $\{\mathcal{A}_\alpha\}$ over which cross-validation was used in order to determine α and obtain the fully specified learner $\mathcal{A}$. When each family member was trained it was done using a training set smaller than S. For k the number of folds used and m the number of samples in S, each model was trained using $m\,(k-1)\,/k$ samples.

Now, recall that successful learning depends on the number of training samples. When discussing PAC theory we defined the sample complexity as the minimal number of samples required to learn an hypothesis from a given hypothesis class, given some ε, δ. If m samples is a sufficient training size, but $m\,(k-1)\,k$ is not, we cannot guarantee the bounds over the generalization error. In such cases we will estimate the generalization error **too high**, namely over-estimate it.

To better determine the number of folds, we would like to have an hypothetical learning curve (Figure 6.9) for the model in question. Such a curve will capture the essence of the sample complexity function. Given a certain number of training samples the curve shows the expected success rate. Using think curve over a training set S with m samples, we can calculate the effective training set size when using k-fold cross-validation for a certain k. If the effective training size remains in the saturated area of the graph the estimated generalization error using cross-validation would not differ by much from the real generalization error. However, if the effective training size is where the slope of the graph is steep, we do not have a sufficient amount of training samples and will suffer from over-estimating the generalization error.
But how to estimate the graph? Maybe add a lab? or maybe add it here?

6.2.4.2 Under-estimating Generalization Error

A much more common mistake is being too optimistic in estimating the generalization error, causing under-estimation of it. Suppose we are given a learning problem and a dataset with m samples. We begin trying different types of models, each with its own tuning parameters. Perhaps we also alter the training sample S by removing- or adding features and addressing "problematic" samples. Eventually we have a "clean" training sample and a tuned model that works well. Now we perform

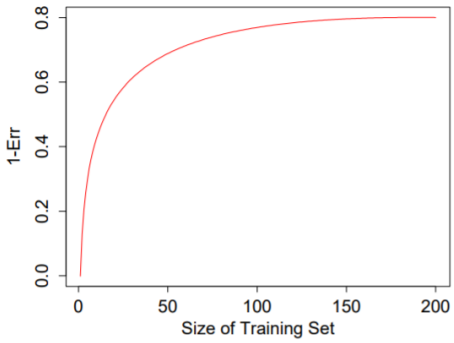


***Figure 6.9: Learning Curve of classifier**: showing the success $(1 - Err)$ as a function of the training set size*

model evaluation to estimate the generalization error of the chosen learner.

In such a scenario we will end up under-estimating the generalization error due to two mistakes:

- The first is known as **model snooping**. By trying out many learners, tuning the parameters of each and looking for one model that will perform very well, we slowly begin overfitting to the training sample. Each time we discarded some potential hypothesis class in favor of another that performed better we introduced more bias to the procedure. Even in the case where we use a validation set, if we evaluate the performance over it many times, we begin overfitting to it as well.
- The second is known as **data snooping**. Once we evaluate our selected model over a new test sample ()or perhaps even when using it in production) this data did not under-go the same level of treatment. It might contain missing features or "problematic" samples that were dealt with in S. Therefore, the cross-validation procedure will under-estimate the generalization error over the new data.

To avoid this problem we should deal with these types of snooping. Limit model- and data-snooping to a small subset of S which will be "contaminated with optimism". Use it to get general understanding of the data and potential models. Only once the snooping stage has completed and a small set of candidate learners is chosen, use the entire training set for model selection- and evaluation. In addition, avoid manual data snooping. **Code the entire pre-processing stage**. At each iteration of Bootstap of cross-validation run the eintire pre-processing step over the current sample, just like it would run when predicting over new data.

6.3 Summary

6.3.1 Lab: Selecting Regularized Model

6.3.2 Lab: Regularized Logistic Regression

7. Unsupervised Learning

Up to this point, our learning paradigm was of **supervised batch learning**. Given a sample space $\mathcal{X}$ and a label/response space $\mathcal{Y}$, we were interested in **prediction**: finding some way to predict $y \in \mathcal{Y}$ corresponding to a new unseen sample $x \in \mathcal{X}$, based on a training set of labeled samples $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$. However, there are many learning problems that do not fit into this framework of supervised batch learning. In one such set of problems we still consider a domain set $\mathcal{X}$ but there is no label/response space. Namely, the data is of the form $S = \{\mathbf{x}_i\}_{i=1}^m$ where $\mathbf{x}_i \in \mathcal{X}$. Such problems are called **un-supervised** learning problems, and the goal is to infer different properties of $\mathcal{X}$. A few examples are:

- **Uncovering low-dimensional structures:** In some cases we have reason to believe that some given data, though represented in high dimension, could actually be represented in a lower dimension. Consider for example the MNIST database of handwritten digits. This coprus of 28-by-28 pixels grey-scaled images shows scans of people handwriting of the digits $0, \ldots, 9$. Though each imange (sample) is represented in $\mathbb{R}^{28\times 28}$, there are very few variations on how people write digits. In Figure 7.1 we can see that in most images a big area in the surrounding of the image remains constant. This means that these pixels are not informative for predicting the digit. In addition, even though two images of the same digit (for example the digit 3 as seen below) show differences, they are still mostly the same. Therefore, it might be possible to **reduce the dimension** of the samples into a more compact, of lower dimension, representation.

- **Clustering:** Often, when we are given a dataset, we are interested in grouping (or segmenting) it into subsets such that those samples within each cluster are in some way more "closely related" to each other than to samples in other subsets. We refer to these subsets as *clusters*.

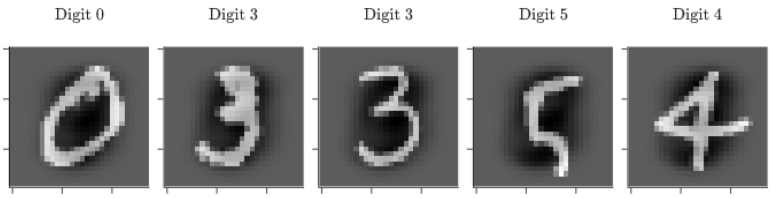


Figure 7.1: MNIST Digits Dataset*: A set of 5 labelled samples from the MNIST Digits dataset*

Consider once more the MNIST digits dataset. In general, it seems logical that images of the same digit resemble each other more than images of different digits. If we would to try to cluster it (that is, segment its samples into different subsets) we would hope to get 10 separate subsets, each containing images of a specific digit.

- **Anomaly detection:** Suppose we are interesting in detecting when some given system is not behaving "as usual". Consider an air conditioning system or power plant. We place sensors in the system and would like to get a warning when something is behaving "strange". We do not know what exactly a "strange" behavior would look like, but it might be caused due to some mechanical malfunction, a software problem or even a cyber attack. We monitor the state of the system at all times. If we have installed d sensors, each time we take a reading of the system we get a sample $\mathbf{x}_i \in \mathbb{R}^d$. We train out learning system on the readings $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ where we believe these represent the normal state of the system. Now, given a new reading $\mathbf{x} \in \mathbb{R}^d$, is it "normal"? or is there something wrong? We are required to make a decision without have seeing the system in the "wrong"/"abnormal"/"strange" state before.

Figure 7.2 shows an example of performing both clustering and dimensionality reduction. In the figure we visualize a dataset of single-cell RNA-sequencing gene expression. This dataset consists of 8850 cells (samples, seen in figure as the individual dots) over 30449 genes (features) where for each cell we have the level of expression (a number in $\mathbb{N}_0$) of the specific gene. The analysis of this dataset included:

- **Dimensionality reduction**: to reduce dimensionality from the original 30449 dimensional features space to a lower 25 dimensional space. Further computations were performed over this lower dimension space, making computations simpler. The dimensionaliy reduction algorithm in use is the PCA algorithm which we discuss below (subsection 7.1.1).

- **Clustering**: Over the reduced, 25 dimensional, feature space a clustering algorithm was used. This algorithm split the dataset into different clusters. These different clusters are represented using different colors.

- **Secondary dimensionality reduction**: To help with visualizing the data, another dimensionality reduction algorithm was used. This time, dimension was reduced to a two dimensinoal space, enabling clear overview of both data and clustering results.

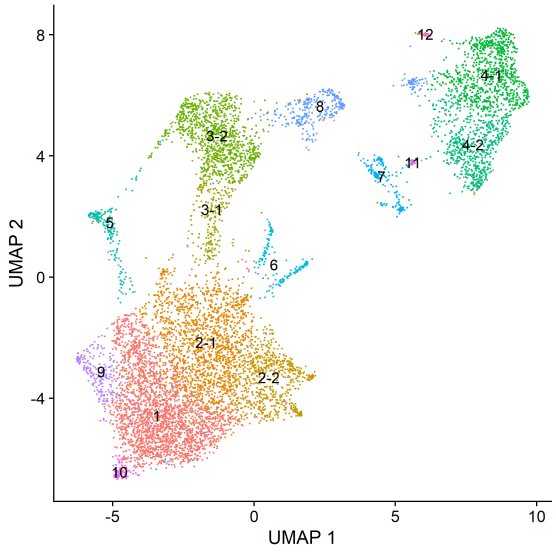


Figure 7.2: Clustering and Dimensionality Reduction

7.1 Dimensionality Reduction

Dimensionality reduction is the process of mapping some high dimensional space (the ambient space) to some new low dimensional space (the intrinsic space). Given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ we want to find some mapping $f : \mathbb{R}^d \to \mathbb{R}^k$ for $k \ll d$, where $f(\mathbf{x}_1), \ldots, f(\mathbf{x}_m)$ still resembles $\mathbf{x}_1, \ldots, \mathbf{x}_m$ in some sense. Different dimensionality reduction techniques are usually separated to linear- and non-linear techniques depending on the selected f.

There are different reasons to study and apply dimension reduction:

- **Learnability:** Some learning algorithms work better when the dimension d is small compared with the training sample size m, and might even fail if d is too large. Recall for example the case of linear regression. Though in both cases $m < d$ and $d < m$ we have a closed form solution, we have seen that if $d < m$ the results are numerically unstable.

- **Computation:** For many algorithm the time- and space complexity depends on the dimension d. By reducing the data dimensionality we can use fewer computational resources to perform the learning task we intended.

- **Visualization:** Visualization is an extremely useful tool both for explaining and exploring our data. If we can reduce dimension to $k \leq 4$ then we can plot the data (possibly on a 3 dimensional axis, with the forth dimension represented by color).

7.1.1 Principal Component Analysis (PCA)

Let $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ be some dataset and suppose that it approximately resides in some lower k-dimensional linear subspace of $\mathbb{R}^d$. We would like to find some linear transformation of the data to project it on that lower dimension subspace. To do so we need to solve two challenges:

1. There are infinitely many subspaces we could consider. Which subspace should we choose

and how should we acquire it?

2. Even if we knew the subspace, projecting the data-points $\mathbf{x} \in \mathbb{R}^d$ onto the subspace does not yet reduce the dimension. We would want to find a way to represent each sample by the k coordinates of that sample *in* the subspace. This is also known as the *embedding* of the samples.

In the case of Principal Component Analysis (PCA), given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^d$ we are searching for some linear transformation such that we minimize the squared error between the original samples and their transformation:

$$f^* := \underset{f}{argmin} \sum_{i=1}^{m} ||\mathbf{x}_i - f(\mathbf{x}_i)||^2 \tag{7.1}$$

This problem can be formulated in four different ways:

1. Finding the closest affine subspace to the points.
2. Finding the affine subspace that retains most of the variation seen in the data (sometimes referred to as *signal*).
3. Finding the affine subspace that minimizes the distortion of the pairwise distances between points in the ambient- compared to the intrinsic spaces.
4. Generalization of linear regression with Gaussian noise both in the explanatory variables directions and in the response direction. This interpretation is also known as probabilistic PCA.

Here, we will discuss the first two, most common, formulations.

7.1.1.1 Closest Affine Subspace

To simply the problem of finding the closest affine subspace, let us assume that the data is centered around the origin and as such instead of searching for some affine subspace, we are searching for an "ordinary" subspace. Therefore, we are searching for some linear map $W : \mathbb{R}^d \to \mathbb{R}^k$ and an "inverse" linear map $U : \mathbb{R}^k \to \mathbb{R}^d$

$$W^*, U^* = \underset{W \in \mathbb{R}^{d \times k}, U \in \mathbb{R}^{k \times d}}{argmin} \sum_{i=1}^{m} ||\mathbf{x}_i - UW\mathbf{x}_i||^2 \tag{7.2}$$

Lemma 7.1.1 Let (U, W) be a solution to (7.2). Then U's columns are orthonormal and $W = U^\top$.

Proof. Let U, W be some matrices and consider the mapping $\mathbf{x} \mapsto (UW)\mathbf{x}$. The matrix (UW) is a d-by-d matrix of rank k, whos image is a k dimensional subspace of $\mathbb{R}^d$. Denote $S := Im(UW)$. As $\mathbf{x} \in \mathbb{R}^d$ and $(UW)\mathbf{x} \in S$, it holds that the projection that minimizes $||\mathbf{x} - UW\mathbf{x}||_2$ is the orthogonal projection onto the subspace. In addition, the point in S closest to $\mathbf{x}$, namely the orgothonal projection pf $\mathbf{x}$ onto S, is given by $VV^\top\mathbf{x}$ where the columns of V are an orthonormal basis of S:

$$\forall \mathbf{u} \in S \quad ||\mathbf{x} - \mathbf{u}||_2 \geq \left|\left|\mathbf{x} - VV^\top\mathbf{x}\right|\right|_2$$

Thus, the solution to 7.2 is U with orthonormal columns and $W := U^\top$. ■

Based on the above lemma, we can now write an equivalent problem to the PCA problem presented in 7.2. Observe that:

$$\begin{aligned}\left|\left|\mathbf{x}-UU^\top\mathbf{x}\right|\right|^2 &= ||\mathbf{x}||^2-2\mathbf{x}^\top UU^\top\mathbf{x}+\mathbf{x}^\top UU^\top UU^\top\mathbf{x}\\ &= ||\mathbf{x}||^2-\mathbf{x}^\top UU^\top\mathbf{x}\\ &\overset{(*)}{=} ||\mathbf{x}||^2-\left(U^\top\mathbf{x}\right)^\top U^\top\mathbf{x}\\ &= ||\mathbf{x}||^2-trace\left(U^\top\mathbf{x}\left(U^\top\mathbf{x}\right)^\top\right)\\ &= ||\mathbf{x}||^2-trace\left(U^\top\mathbf{x}\mathbf{x}^\top U\right)\end{aligned}$$

where $(*)$ is because $\forall\mathbf{v},\mathbf{u}\in\mathbb{R}^k\ trace\left(\mathbf{u}\mathbf{v}^\top\right)=\mathbf{v}^\top\mathbf{u}$. As the trace is a linear operator we can re-write an equivalent problem:

$$U^*=\underset{U\in\mathbb{R}^{d\times k},U^\top U=I}{argmax}\sum_{i=1}^m trace\left(U^\top\mathbf{x}_i\mathbf{x}_i^\top U\right)=\underset{U\in\mathbb{R}^{d\times k},U^\top U=I}{argmax}\ trace\left(U^\top\sum_{i=1}^m\mathbf{x}_i\mathbf{x}_i^\top U\right)\tag{7.3}$$

Theorem 7.1.2 Let $A=\sum\mathbf{x}_i\mathbf{x}_i^\top$ and let $\mathbf{u}_1,\ldots,\mathbf{u}_k$ be the k leading eigenvectors of A. Then, the solution to the PCA problem is given by $U\in\mathbb{R}^{d\times k}$ whose columns are $\mathbf{u}_1,\ldots,\mathbf{u}_k$.

Proof. Denote $A=\sum\mathbf{x}_i\mathbf{x}_i^\top$. Then, we need to solve

$$\underset{U\in\mathbb{R}^{d\times k},U^\top U=I}{argmax}\ trace\left(U^\top AU\right)$$

Notice that A is a square symmetric matrix so let $A=VDV^\top$ be the EVD of A, where the diagonal of D are the eigenvalues of A in decreasing order $\lambda_1\geq\ldots\geq\lambda_d$ and the columns of V are the corresponding eigenvectors. So:

$$\begin{aligned}trace\left(U^\top AU\right) &= trace\left(U^\top VDV^\top U\right) &\overset{(*)}{=}\ & trace\left(B^\top DB\right)\\ &= trace\left(DBB^\top\right) &=\ & \textstyle\sum_{j=1}^d\left[DBB^\top\right]_{jj}\\ &= \textstyle\sum_{j=1}^d[D]_j\left[BB^\top\right]_{\cdot,j} &=\ & \textstyle\sum_{j=1}^d\lambda_j\cdot\left[BB^\top\right]_{jj}\\ &= \textstyle\sum_{j=1}^d\lambda_j\cdot\sum_{i=1}^k B_{ji}^2 &&\end{aligned}$$

where $B=V^\top U\in\mathbb{R}^{d\times k}$. Notice that

$$\begin{gathered}B^\top B=U^\top VV^\top U=I_d\\ \Downarrow\\ \mathbb{1}[j\neq i]=\left\langle\left[B^\top\right]_j,[B]_{\cdot,i}\right\rangle=\left\langle[B]_{\cdot,j},[B]_{\cdot,i}\right\rangle\end{gathered}$$

meaning that the columns of B are orthonormal, and therefore $\sum_{j=1}^d\sum_{i=1}^k B_{ji}^2=k$. Based on B, define $\widetilde{B}=\left[\begin{array}{c|c}B & M\end{array}\right]\in\mathbb{R}^{d\times d}$ with $M\in\mathbb{R}^{d\times(d-k)}$ completing an orthonormal basis in R^d. If so, then $\forall j\ \sum_{i=1}^d\widetilde{B}_{ji}^2=1$, which implies that $\sum_{i=1}^k B_{ji}^2$. As such

$$trace\left(U^\top AU\right)=\sum_{j=1}^d\lambda_j\cdot\sum_{i=1}^k B_{ji}^2\leq\underset{\beta\in[0,1]^d,||\beta||_1\leq k}{max}\sum_{j=1}^d\lambda_j\beta_j=\sum_{j=1}^k\lambda_k$$

To conclude the proof, let U be the matrix whose columns are the k leading eigenvalues of A then

$$U^\top AU = \begin{bmatrix} u_1^\top \\ \vdots \\ u_k^\top \end{bmatrix} A \begin{bmatrix} u_1 & \cdots & u_k \end{bmatrix} = \begin{bmatrix} u_1^\top \\ \vdots \\ u_k^\top \end{bmatrix} \begin{bmatrix} \lambda_1 u_1 & \cdots & \lambda_k u_k \end{bmatrix} = diag\left(\lambda_1, \ldots, \lambda_k, \overbrace{0, \ldots, 0}^{d-k}\right)$$

and $trace\left(U^\top AU\right) = trace\left(diag\left(\lambda_1, \ldots, \lambda_k, 0, \ldots, 0\right)\right) = \sum^k \lambda_i$, as requested. ■

Generalizing for affine subspaces:
In the above theorem we in-fact found the closest subspace- and not the closest affine subspace to the data. To find the closest affine subspace, we would like to allow the mapping W to be **affine**. To achieve this we generalize the above by considering $W : \mathbb{R}^d \to \mathbb{R}^k$ to be of the form

$$W(\mathbf{x}) := \widetilde{W}(\mathbf{x} - \mu), \quad \mu \in \mathbb{R}^d, \widetilde{W} : \mathbb{R}^d \to \mathbb{R}^k$$

This allows us to "shift" the data before applying the linear map $\widetilde{W}$. When adding μ to the optimization problem above, we find that the minimizer μ is given by: $\mu = \frac{1}{m}\sum \mathbf{x}_i$. This is the empirical mean of the data, often denoted by $\overline{\mathbf{x}}$. So, in order to find the closest affine subspace to the data we center the matrix A:

$$A := \sum_{i=1}^m (\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^\top$$

yielding the following pseudo-code for the PCA algorithm:

Algorithm 2 PCA

procedure PCA($\mathbf{X}$, k) ▷ $\mathbf{X}$ The design matrix of m samples and d features
  Compute $A \leftarrow \sum_{i=1}^m (\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^\top$
  Let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ be the eigenvectors of A corresponding the largest eigenvalues.
  return $\mathbf{u}_1, \ldots, \mathbf{u}_k$
end procedure

The eigenvalues of A, $\lambda_1 \geq \ldots \geq \lambda_d$ are referred to as the **Principal Values** of $\mathbf{X}$. The eigenvectors of A, $\mathbf{u}_1, \ldots, \mathbf{u}_d$ are referred to as the **Principal Components**. Notice that the matrix A is a d-by-d positive semi-definite matrix. Therefore the PCA algorithm is in fact a case of a matrix diagonalization algorithm where we simply try to find the eigenvalues and eigenvectors of some target matrix.

7.1.1.2 Maximum Retained Variance

Another way to think about PCA is as a dimensionality reduction technique that maintains the maximum amount of variance of the data possible in a $k < d$ dimensional subspace. To prove so we will have to solve a constraint maximization problem using Lagrange Multipliers.
Explain lagrange multipliers

Theorem 7.1.3 Let $\mathbf{X}$ be an m-by-d design matrix. The projection of $\mathbf{X}$ onto a k dimensional linear subspace that retains maximum of the variance in $\mathbf{X}$ is given by the matrix $U \in \mathbb{R}^{d\times k}$ whose columns are the k eigenvectors with leading eigenvalues of the sample covariance matrix S.

Proof. We begin with considering the projection onto a one-dimensional subspace. Without loss of generality let $\mathbf{v} \in \mathbb{R}^d$ be a unit vector used to project the data onto. The variance of the projection of each $\mathbf{x}_i$ onto $\mathbf{v}$, $\mathbf{v}^\top \mathbf{x}_i$ is given by:

$$\begin{aligned}
\mathbb{E}\left[\mathbf{v}^\top \mathbf{x}\right] &= \tfrac{1}{m}\sum \mathbf{v}^\top \mathbf{x}_i = \mathbf{v}^\top \overline{\mathbf{x}} \\
Var\left(\mathbf{v}^\top \mathbf{x}\right) &= \mathbb{E}_{\mathbf{x}}\left[\left(\mathbf{v}^\top \mathbf{x}_i - \mathbb{E}_{\mathbf{x}}\left[\mathbf{v}^\top \mathbf{x}\right]\right)^2\right] = \tfrac{1}{m}\sum\left(\mathbf{v}^\top \mathbf{x}_i - \mathbf{v}^\top \overline{\mathbf{x}}\right)^2 \\
&= \tfrac{1}{m}\sum\left[\mathbf{v}^\top\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)\right]^2 = \tfrac{1}{m}\sum\left[\mathbf{v}^\top\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)\right]\left[\mathbf{v}^\top\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)\right]^\top \\
&= \tfrac{1}{m}\sum \mathbf{v}^\top\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)^\top \mathbf{v} = \mathbf{v}^\top\left[\tfrac{1}{m}\sum\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)\left(\mathbf{x}_i - \overline{\mathbf{x}}\right)^\top\right]\mathbf{v} \\
&= \mathbf{v}^\top S\mathbf{v}
\end{aligned}$$

So now let us maximize the projected variance with respect to $\mathbf{v}$:

$$\widehat{\mathbf{v}} = \underset{||v||=1}{argmax}\, \mathbf{v}^\top S\mathbf{v}$$

To solve this optimization problem we will use the Lagrange multipliers method with the constraint $g(\mathbf{v}) = 1 - \mathbf{v}^\top\mathbf{v}$. So the lagrangian is given by:

$$\begin{gathered}
\mathcal{L} = \mathbf{v}^\top S\mathbf{v} + \lambda g(\mathbf{v}) \\
\Downarrow \\
\tfrac{\partial}{\partial \mathbf{v}}\mathcal{L} = 2S\mathbf{v} - 2\lambda\mathbf{v} = 0
\end{gathered}$$

Therefore the maximizer of $\mathbf{v}^\top\mathbf{v}$ must be an eigenvector of S. Notice that by left-multiplying the derivative by $\mathbf{v}^\top$ we find that the retained variance itself is given by:

$$\mathbf{v}^\top S\mathbf{v} = \lambda\mathbf{v}^\top\mathbf{v} \overset{||\mathbf{v}||=1}{=} \lambda$$

Thus, the maximal retained variance is the largest eigenvalue λ_1, achieved by $\mathbf{v} := \mathbf{u}_1$.

Next, let us find the direction that retains the second largest amount of variance. As we are looking for an orthogonal projection we add an additional constraint that this direction is orthogonal to $\mathbf{u}_1$:

$$\widehat{\mathbf{v}} = \underset{||\mathbf{v}||=1,\, \mathbf{v}^\top\mathbf{u}_1=0}{argmax}\ \mathbf{v}^\top S\mathbf{v}$$

As before, when solving the constraint optimization problem for $\mathbf{v}$ we find that the direction retaining the second largest amount of variance is $\mathbf{v} := \mathbf{u}_2$, with the amount of variance being λ_2. Proving by induction we get that $\forall k \leq d$, the k dimensional subspace that retains maximal variance of projecting $\mathbf{X}$ onto it, is given by the k leading eigenvectors of S. ■

7.1.1.3 Link Between Closest Subspace and Maximum Variance

In the above we have seen two interpretations of PCA: one as closest subspace and one as maximizing variance. To understand how the two are connected consider the dataset seen in Figure 7.3 and specifically the $\mathbf{x}_i$ data-point. Notice that by orthogonally projecting $\mathbf{x}_i$ onto $\mathbf{u}_1$ a right-angle triangle is formed where:

- The edge denoted by a is the size of the projection of $\mathbf{x}_i$ onto $\mathbf{u}_1$: $a \coloneqq \left|\left|\mathbf{x}_i^\top \mathbf{u}_1\right|\right|$. This is the measure maximized in the maximum variance interpretation.
- The edge denoted by b is the distance between the original data-points $\mathbf{x}_i$ and its orthogonal projection onto $\mathbf{u}_1$: $b \coloneqq ||\mathbf{x}_i - \mathbf{x}_i\mathbf{u}_1||$. This is the measure minimized in the closest subspace interpretation.
- The edge denoted by c is the size of $\mathbf{x}_i$: $c \coloneqq ||\mathbf{x}_i||$.

As this is a right-angle triangle, we know from the Pythagorean theorem that $c^2 = a^2 + b^2$. Therefore, if we find a PCA solution that minimizes b (the closest subspace), we in fact find a solution that maximizes a. Similarly by finding a solution that maximizes a (maximal projected variance), we find a solution that minimizes b.

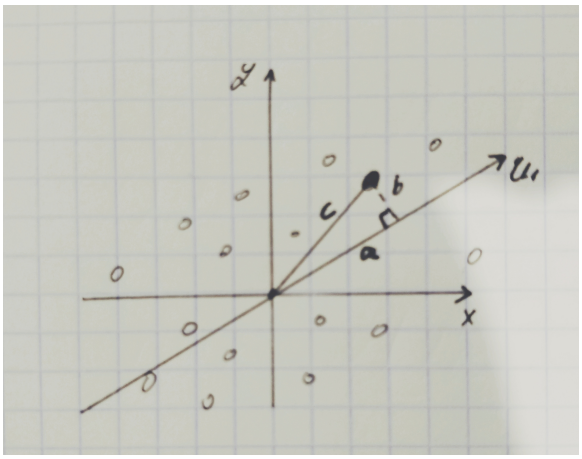


***Figure 7.3: Link Between PCA Interpretations**: Projection of $\mathbf{x}_i$ onto $\mathbf{u}_1$ forming a right-angle triangle.*

7.1.1.4 Projection- vs. Coordinates of Data-Points

When considering PCA (or many other dimensionality reduction algorithms), an important but often overlooked point is the difference between projection and embedding (i.e coordinates) of the data-points. Suppose $\mathbf{X} \in \mathbb{R}^{m \times d}$ and we run the PCA algorithm to find a lower k dimensional linear subspace. The optimal PCA solution is the subspace that minimizes the sum of squared distances between each data-point $\mathbf{x}_i$ and its orthogonal projection onto the subspace. As we have seen above (7.1.2, 7.1.3), this subspace is spanned by the k leading eigenvectors of the $d \times d$ sample covariance matrix $S = \frac{1}{m}\sum (\mathbf{x}_i - \bar{x})(\mathbf{x}_i - \bar{x})^\top$.

This matrix $U \in \mathbb{R}^{d \times k}$ enables the **projection** of the points in $\mathbb{R}^d$ onto the k dimensional subspace, spanned by these leading eigenvectors. However, we would also like to actually reduce the dimension. Namely, find the map $W : \mathbb{R}^d \to \mathbb{R}^k$ and work with the dimension-reduced dataset $W(\mathbf{x}_1), \ldots, W(\mathbf{w}_m)$. As proven above, it is in fact $W = U^\top$ the map that provides the **embedding** of the data into the low

dimension space. The vector $U^\top \mathbf{x}_i$ is a k dimensional vector, laying within the found k dimensional subspace. These are the **coordinates** of the original vector $\mathbf{x}_i$ according to the orthonormal set of k leading eigenvalues of S.

Let $\mathbf{u}_1, \ldots, \mathbf{u}_d$ be the d eigenvectors of S numbered in ascending order by the associated eigenvalues. As these vectors form a basis in $\mathbb{R}^d$, the vector $\mathbf{x}_i$ can be decomposed as $\mathbf{x}_i = \sum_{j=1}^{d} \langle \mathbf{x}_i, \mathbf{u}_j \rangle \mathbf{u}_j$.

- The **projecting** of $\mathbf{x}_i$ on the optimal k-dimensional subspace is given by $\mathbf{x}_i = \sum_{j=1}^{k} \langle \mathbf{x}_i, \mathbf{u}_j \rangle \mathbf{u}_j$.
- The **coordinates** (i.e. embedding) of $\mathbf{x}_i$ according to the k leading principal vectors are given by: $\left(\langle \mathbf{x}_i, \mathbf{u}_1 \rangle, \ldots, \langle \mathbf{x}_i, \mathbf{u}_k \rangle\right)^\top$.

In Figure 7.4 we illustrate the difference between the projection and the coordinates (embedding) of the data-points. This dataset was generated as follows: 1000 points were sampled from the ℓ_2 unit ball in $\mathbb{R}^2$. That is, $\{((x_i)_1, (x_i)_2)\}_{i=1}^{1000}$ where $(x_i)_1^2 + (x_i)_2^2 = 1$. Then Gaussian noise was added in a third axis:

$$\{((x_i)_1, (x_i)_2, \varepsilon_i)\}_{i=1}^{1000} \quad (x_i)_1^2 + (x_i)_2^2 = 1 \quad \varepsilon_1, \ldots, \varepsilon_{1000} \overset{iid}{\sim} \mathcal{N}(0, 0.1)$$

Figure 7.4 (left) shows this dataset. Then, using PCA we first project the data onto the subspace defined by the 3 PCs (Figure 7.4, center). At this point each data-point is still represented in $\mathbb{R}^3$, but we can indeed see that it is mainly the first 2 axes that capture the signal in the data, with the third axis showing only minor variations (the added noise). Lastly, when embedding the data into a 2 dimensional subspace (Figure 7.4, right), we are left with the true signal of the data (i.e. samples drawn from a 2 dimensional ℓ_2 unit ball), and data-points are represented using only 2 coordinates.

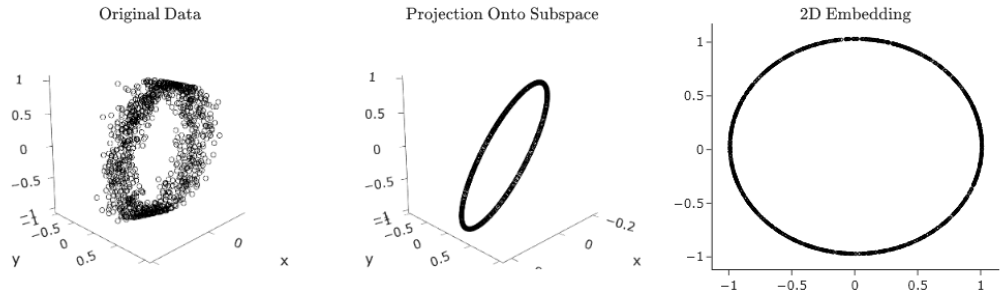


Figure 7.4: Projection vs. Embedding*: Dataset of sampels taken on the ℓ_2 unit ball with Gaussian noise in $\mathbb{R}^3$, and then rotated in a random direction. Chapter 7 Examples - PCA - Source Code*

7.1.1.5 Principal Components As "Typical Data-Points"

The principal components found by PCA are vectors in the ambient space $\mathbb{R}^d$. This means that they share the same dimension as the data-points $\mathbf{x}_1, \ldots, \mathbf{x}_m$. As they are orthonormal vectors chosen such that the first k provide the best linear approximation of dimension k to the dataset, we can look at them in an interesting way. They are, in this sense, the "typical" data-points, maximally different from each other (as they are an orthonormal set). Thus, it is often interesting to see what they represent as data-points: what part of the "signal" do they capture.

Returning to the MNIST Digits dataset of Figure 7.1 we can look at each principal component as a

28-by-28 image. Then, we can think of the representation of each image by the linear combination of these images:

$$\widehat{f}(\mathbf{x}) = \sum_{i=1}^{k} \alpha_i \mathbf{u}_i + \overline{\mathbf{x}}, \quad \alpha_1, \ldots, \alpha_k \in \mathbb{R}$$

where $\mathbf{u}_1, \ldots, \mathbf{u}_k$ are the leading k principal components and $\widehat{\mathbf{x}}$ is the centering vector of the data. Notice that as this is a linear combination we are reconstructing the sample $\mathbf{x}$ via adding and subtracting (weighted) principal components.

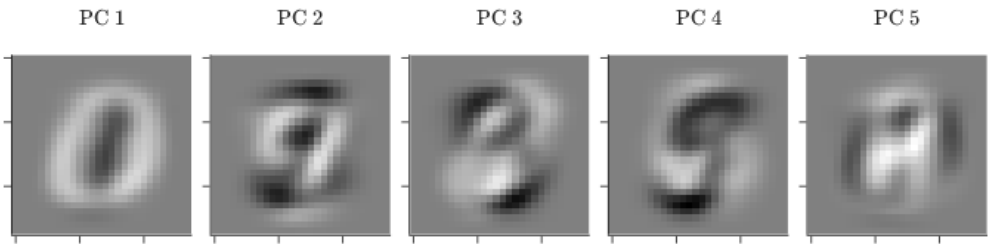


Figure 7.5: Top Principal Components *of PCA fitted over the MNIST Digits dataset.* *Chapter 7 Examples - PCA - Source Code*

Figure 7.6 shows the reconstruction process of an image of digit 4 as the linear combination of the principal components. In each frame of the animation the restored image (center) is calculated as the restored image of the previous iteration plus $\alpha_i \mathbf{u}_i$ for $\mathbf{u}_i$ the principal component of the current iteration and α_i the loadings (coordinates) of the image for the i'th principal component.

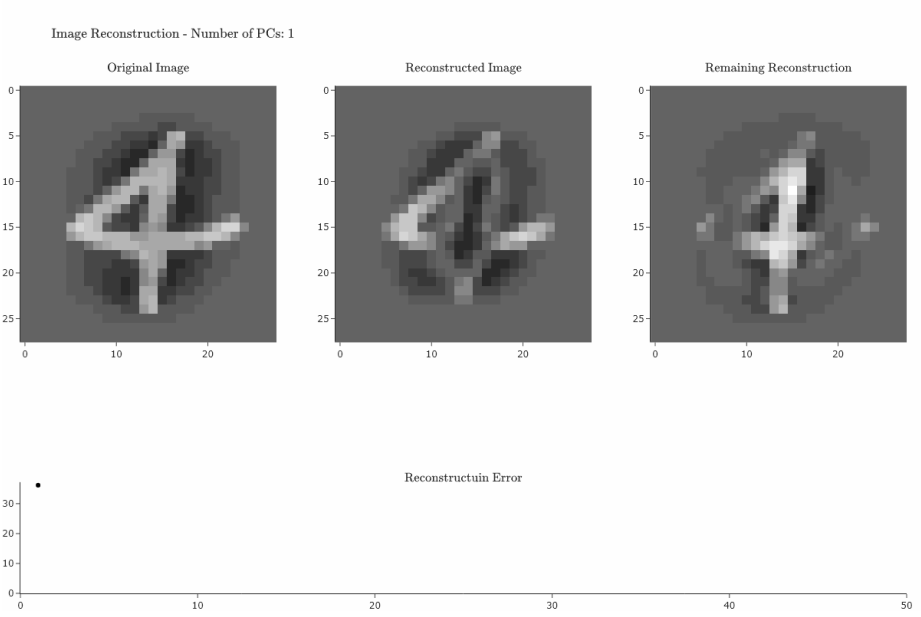


Figure 7.6: PCA Reconstruction: *Constructing image from principal components.* *Chapter 7 Examples - PCA - Source Code*

7.1.2 Kernel PCA

The PCA algorithm discussed above is sometimes referred to as a "Vanilla" PCA, with many different variations of the algorithm to fit different scenarios. For one such variation, we apply kernel substitution techniques to obtain a **non-linear generalization** of PCA known as Kernel PCA (Schölkopf *et al.,* 1998).

Recall the general setup of Kernel methods: Suppose some algorithm $\mathcal{A}$ and let $\mathbf{X} \subset \mathcal{X}$ be an m-by-d design matrix. For some mapping function $\phi : \mathcal{X} \to \mathcal{F}$, we first apply $\phi(\mathbf{X}) := \{\phi(\mathbf{x}) \,|\mathbf{x} \in \mathbf{X}\}$ and then run $\mathcal{A}$ over $\phi(\mathbf{X})$. By selecting ϕ is a smart manner we might be able to achieve a mapping to:

- A more informative feature space, where our algorithm will perform better (recall the use in Soft-SVM for non-separable datases add reference).
- A new feature space of a much higher dimension (perhaps even of infinite dimension) while still being able to efficiently compute the transformation (i.e. ϕ a valid kernel function and use of the Kernel Trick).

To apply the Kernel Trick on an algorithm $\mathcal{A}$ we require the algorithm to never access the data $\mathbf{x} \in \mathbf{X}$ directly, but only though the use of inner-products. Thus, in order to efficiently perform such kernel transformation and achieve a Kernel PCA algorithm, we must first show that we can re-write PCA to access the data only though inner-products.

Claim 7.1.4 Let $\mathbf{X}$ be a m-by-d centered design matrix. Solving the PCA eigenvalue problem for $A = \mathbf{X}^\top \mathbf{X}$ is equivalent to solving the eigenvalue problem for $K = \mathbf{X}\mathbf{X}^\top$.

Proof. In PCA we solve the eigenvalue problem of the matrix $A = \mathbf{X}^\top \mathbf{X} = \sum \mathbf{x}_i \mathbf{x}_i^\top$. We would like to represent this sum of outer-products via inner-products. Let v be an eigenvector of A corresponding eigenvalue λ then:

$$\begin{gathered} \lambda v = Av = \sum \mathbf{x}_i \mathbf{x}_i^\top v = \sum \langle \mathbf{x}_i, v \rangle \mathbf{x}_i \\ \Downarrow \\ v = \sum \tfrac{1}{\lambda} \langle \mathbf{x}_i, v \rangle x_i \end{gathered}$$

This means that solving an eigenvalue problem for A is equivalent to finding a vector v such that

$$\lambda \langle \mathbf{x}_i, v \rangle = \langle \mathbf{x}_i, Av \rangle \quad i = 1, \ldots, m$$

Fix $i \in [m]$ and denote $\alpha_i = \frac{1}{\lambda} \langle \mathbf{x}_i, v \rangle$, so:

$$\begin{aligned} \lambda \left\langle \mathbf{x}_i, \textstyle\sum_j \alpha_j \mathbf{x}_j \right\rangle &= \left\langle \mathbf{x}_i, A \left(\textstyle\sum_j \alpha_j \mathbf{x}_j \right) \right\rangle \\ &= \\ &= \textstyle\sum_j \alpha_j \left\langle \mathbf{x}_i, A \mathbf{x}_j \right\rangle \\ &= \textstyle\sum_j \alpha_j \left\langle \mathbf{x}_i, \sum_l \mathbf{x}_l \mathbf{x}_l^\top \mathbf{x}_j \right\rangle \\ &= \textstyle\sum_j \alpha_j \sum_l \left\langle \mathbf{x}_l, \mathbf{x}_j \right\rangle \mathbf{x}_l \\ &= \textstyle\sum_j \alpha_j \sum_l \left\langle \mathbf{x}_l, \mathbf{x}_j \right\rangle \left\langle \mathbf{x}_i, \mathbf{x}_l \right\rangle \\ &\Downarrow \\ \lambda \textstyle\sum_j \alpha_j \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle &= \textstyle\sum_j \alpha_j \sum_l \left\langle \mathbf{x}_l, \mathbf{x}_j \right\rangle \left\langle \mathbf{x}_i, \mathbf{x}_l \right\rangle \end{aligned} \tag{7.4}$$

Which in matrix notation is $\lambda K\alpha = K^2\alpha$ where we define K to be the Gram matrix over $\mathbf{x}_1,\dots,\mathbf{x}_m$: $K_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$, or equivalently $K = \mathbf{X}\mathbf{X}^\top$. For eigenvectors of K, not corresponding to zero eigenvalues the solution of $\lambda K\alpha = K^2\alpha$ is equivalent to that of $\lambda\alpha = K\alpha$. ■

Using the above claim, once we find $\alpha^{(1)},\dots,\alpha^{(m)}$ the eigenvectors of K we can then:

- Obtain the eigenvectors of A by $v^{(l)} = \sum_{i=1}^m \alpha_i^{(l)}\mathbf{x}_i$.
- Project the data-points on the low dimension subspace by:

$$\widetilde{\mathbf{x}}_l := \left\langle v^{(l)}, \mathbf{x} \right\rangle = \sum_i \alpha^{(l)} \left\langle \mathbf{x}_i, \mathbf{x} \right\rangle \tag{7.5}$$

Notice that by showingg that we can calculate the eigenvalues 7.4 and the projection 7.5 using the data only though inner-products we have shown that the Kernel Trick is applicable for the PCA algorithm. Thus, for a kernel function ϕ where $A = \sum \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top$:

- Solve the eigenvalue problem $\lambda\alpha = K\alpha$, for $K_{ij} = \langle \phi(x_i), \phi(\mathbf{x}_i)\rangle$.
- Project the data-points by:

$$\widetilde{\mathbf{x}}_l := \left\langle v^{(l)}, \phi(\mathbf{x}) \right\rangle = \sum_i \alpha^{(l)} \left\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}) \right\rangle = \sum_i \alpha^{(l)} k(\mathbf{x}_i, \mathbf{x})$$

The above claim assumes the given design matrix $\mathbf{X}$ is centered. Unlike PCA, we cannot compute the mean and reduce it as it would require to compute $\phi(\mathbf{X})$. Therefore, we would need to use a different approach:

Lemma 7.1.5 Let K be the Gram matrix of ϕ over the dataset $\mathbf{X}$. The centered Gram matrix to be used in the Kernel PCA algorithm is given by:

$$\widetilde{K} = K - \mathbf{1}_m K - K\mathbf{1}_m + \mathbf{1}_m K \mathbf{1}_m$$

Proof. Denote $\widetilde{\phi}(\mathbf{x})$ the projected data-point after centerlizing:

$$\widetilde{\phi}(\mathbf{x}) = \phi(\mathbf{x}) - \frac{1}{m}\sum_l \phi(\mathbf{x}_l)$$

where $\mathbf{1}_m \in \mathbb{R}^{m\times m}$, $[\mathbf{1}_m]_{ij} = 1/m$. So the elements of the centered Gram matrix are given by:

$$\begin{aligned}
\widetilde{K}_{ij} &= \left\langle \widetilde{\phi}(\mathbf{x}_i), \widetilde{\phi}(\mathbf{x}-j) \right\rangle \\
&= \left\langle \phi(\mathbf{x}_i) - \tfrac{1}{m}\textstyle\sum_l \phi(\mathbf{x}_l), \phi(\mathbf{x}_j) - \tfrac{1}{m}\sum_k \phi(\mathbf{x}_k) \right\rangle \\
&= \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) - \tfrac{1}{m}\textstyle\sum_k \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_k) - \tfrac{1}{m}\sum_l \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_j) + \tfrac{1}{m^2}\sum_{l,k} \phi(\mathbf{x}_l)^\top \phi(\mathbf{x}_k) \\
&= K_{ij} - \tfrac{1}{m}\textstyle\sum_l K_{il} - \tfrac{1}{m^2}\sum_k K_{jk} + \tfrac{1}{m}\sum_{l,k} K_{lk}
\end{aligned}$$

which in matrix notation is $\widetilde{\phi}(\mathbf{x}) = \phi(\mathbf{x}) - \frac{1}{m}\sum_l \phi(\mathbf{x}_l)$. ■

So a pseudo-code for the Kernel PCA algorithm is:

Algorithm 3 Kernel-PCA

procedure KERNEL-PCA($\mathbf{X}$, k, l) ▷ k the kernel computing ϕ and l the dimension to reduce to
Compute the centered Gram matrix (7.1.5):

$$\widetilde{K} = K - \mathbf{1}_m K - K\mathbf{1}_m + \mathbf{1}_m K \mathbf{1}_m, \quad K_{ij} := k(\mathbf{x}_i, \mathbf{x}_j)$$

Let $\alpha^{(1)}, \ldots, \alpha^{(l)}$ be the eigenvectors of $\widetilde{K}$ corresponding the largest eigenvalues.
Compute the corresponding eigenvectors of A by: $\mathbf{v}^{(j)} := \sum_{i=1}^{m} \alpha_i^{(j)} \mathbf{x}_i$
return $\mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(l)}$
end procedure

R The PCA algorithm seen in Algorithm 2 diagonalizes the sample covariance matrix, which is a d-by-d matrix and has a time complexity of $\mathcal{O}(d^3)$. If $d \gg m$ this can be computationally expensive. Though the proof of the Kernel PCA algorithm we have actually shown that we can solve the "normal" PCA by computing $\mathbf{X}\mathbf{X}^\top$ instead of using $\mathbf{X}^\top\mathbf{X}$. This means that time complexity is reduced to $\mathcal{O}(m^3)$ for computing $\mathbf{X}\mathbf{X}^\top$ and then $\mathcal{O}(m^2 \cdot d)$ for adjusting the eigenvalues of $\mathbf{X}\mathbf{X}^\top$ to those of $\mathbf{X}^\top\mathbf{X}$. So we can update the PCA pseudo code as follows:

Algorithm 4 PCA

procedure PCA($\mathbf{X}$, k) ▷ $\mathbf{X}$ The design matrix of m samples and d features
if $m > d$ **then**
Compute $A \leftarrow \sum_{i=1}^{m} (\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^\top$
Let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ be the eigenvectors of A corresponding the largest eigenvalues.
return $\mathbf{u}_1, \ldots, \mathbf{u}_k$
else
Compute $B \leftarrow \mathbf{X}\mathbf{X}^\top$ write in format as of A
Let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be the eigenvectors of B corresponding the largest eigenvalues.
Compute the eigenvectors of A by: $\mathbf{u}_i = \frac{1}{||\mathbf{X}^\top \mathbf{v}_i|| \mathbf{X}^\top \mathbf{v}_i} \quad i = 1, \ldots, k$
return $\mathbf{v}_1, \ldots, \mathbf{v}_k$
end if
end procedure

Add exercise 12.26 in Bishop to explain why could remove K

7.2 Clustering

A very useful set of learning problems is of clustering. Often, either as part of data exploration or as part of the main analysis, we are interested in partitioning our data into **meaningful** groups. For example, given a corpus of images we might want to divide them into different groups such as nature/urban/people/etc. photographs; or, given the set of genes in the human genome, we might want to group together genes associated with specific diseases. In both these examples we are only given the samples (images or genes) but we do not have any **ground truth** (labels). We are not given the information of what is seen in each image, nor for each disease what genes are associated with it.

We would therefore want to define some measure of *similarity* between samples of a given domain. Using this similarity we could split the data into different subsets, where intuitively samples within a given set are *more similar* to one another compared to samples of different sets.

R This form of clustering, where we partition the data into distinct non-overlapping groups is also referred to as "hard assignment" as each sample is assigned to one specific subset. Sometimes, rather than assigning to some specific cluster we are interested in "partly assigning" to multiple clusters. This is referred to as "soft assignment/clustering".

7.2.1 K-Means

Though there are many different approaches to perform clustering, common to many is the notion of defining some representative data-point of the cluster. Then, clustering of data-points give the given sample is perform with respect to these cluster representatives.

Definition 7.2.1 A partition of a dataset $\{\mathbf{x}_i\}_{i=1}^m$ is a set $C_1,\ldots,C_k$ such that $\{\mathbf{x}_i\}_{i=1}^m = \biguplus_{j=1}^k C_j$

Given some partition over the data and the representative data-points, we can define a **cost function** for the partition. For some metric (i.e. distance function) over the data $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ we define:

$$G_d(C_1,\ldots,C_k,\mu_1,\ldots,\mu_k) \coloneqq \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j} d(\mathbf{x},\mu_j) \tag{7.6}$$

where $\mu_1,\ldots,\mu_k$ are the representatives of clusters $C_1,\ldots,C_k$. Using this function, the goal is to find the partitioning that minimizes the following:

$$\{C_1,\ldots,C_k\}^* = \underset{\{C_j,\mu_j\}_{j=1}^k}{argmin}\ G_d(C_1,\ldots,C_k,\mu_1,\mu_k) = \underset{\{C_j,\mu_j\}_{j=1}^k}{argmin} \sum_{j=1}^{k} \sum_{\mathbf{x}\in C_j} d(\mathbf{x},\mu_j) \tag{7.7}$$

In the case of the k-means algorithm the cluster representatives are chosen to be:

$$\mu_j(C_j) \coloneqq \underset{\mu\in\mathcal{X}}{argmin} \sum_{\mathbf{x}\in C_j} d(\mathbf{x},\mu)$$

In order to find the minimizers of 7.7 we would have to navigate the space of all possible partitions of m objects into k subsets. As there are an exponential amount of such subsets, minimizing the cost function G is NP-hard, and we must resort to **heuristics**. The most famous heuristic for minimizing G, when d is the Euclidean distance, is k-means.

This clustering approach uses **Lloyd's Algorithm** to minimize G using an iterative strategy where each time we **alternate** between minimizing two terms. We first define a partition of the dataset, associating points to subsets by their nearest centroid. These subsets are called Voronoi cells. Then we re-calculate the cluster's centroid.

7.2.1.1 Convergence to Multiple- and Sub- Optimal Solutions

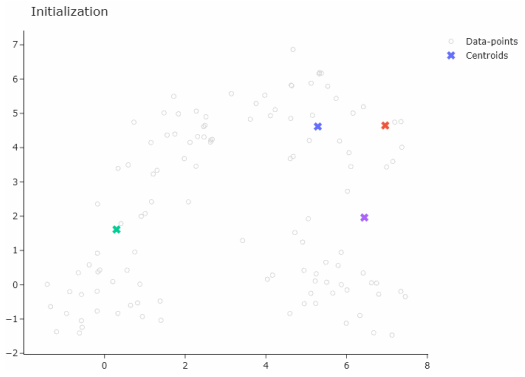
As clustering problems are NP-Hard, the K-Means algorithm algorithm uses the Lloyd's algorithm heuristic to find a good partition of the data. Being a heuristic, the optimality of the algorithm

Algorithm 5 K-Means

procedure K-MEANS($\mathbf{X}$, k) ▷ $\mathbf{X}$ The design matrix of m samples and d features
  Choose initial centroids $\mu_1, \ldots, \mu_k$ randomly.
  while Not converged **do**
    Assignment: Assign each point $\mathbf{x}_i$ to the centroid closest to it:

$$C_j^{(t)} := \left\{ \mathbf{x} | \mu_j = argmin_\mu \, d\left(\mathbf{x}, \mu\right), \mathbf{x} \in \mathbf{X} \right\}$$

    Update: Adjust cluster centroids by: $\forall j \in [k] \quad \mu_j^{(t+1)} := \left| C_j^{(t)} \right|^{-1} \sum_{\mathbf{x} \in C_j^{(t)}} \mathbf{x}$
  end while
  return $C_1, \ldots, C_k$
end procedure



assignment to clusters isn't guaranteed. In fact, due to the nature of the random initialization of centroids, the K-Means algorithm might converge into a sub-optimal solution or to there exists more than a single possible optimal solution.

Sub-optimality means that given a dataset $\{\mathbf{x}_i\}_{i=1}^m$ the K-Means algorithm returned some partitioning $C_1, \ldots, C_k$ such that there exists a different partitioning of the data $C'_1, \ldots, C'_k$ with a lower objective value: $G\left(C_1, \ldots, C_k\right) > G\left(C'_1, \ldots, C'_k\right)$. This is the product of the objective function not being convex, which means there could be several local minima, each achieving a different cost. In Figure 7.7 we see 4 groups of data-points and two initial centroids, positioned in such a way that forces the algorithm to converge into a sub-optimal solution. Optimal assignment is achieved for final centroids $\mu_i = (0,5), \mu_j = (20,5), \; i \neq j$.
Multiple optimal solution means that given a dataset $\{\mathbf{x}_i\}_{i=1}^m$ there are more than a single possible partitioning of the data that will yield the lowest objective value. It is important to note, that whenever discussing clustering assignments, we always look at the partitioning up to a permutation of the partition names. That is, suppose we are given the samples $1,2,3,4,5$ the partitioning of $C_1 = \{1,2,3\}, C_2 = \{4,5\}$ is identical to $C_1 = \{4,5\}, C_2 = \{1,2,3\}$, and clearly, both will achieve the same objective value. When referring to multiple optimal solution we mean:

$$\begin{array}{ll} \textit{Given } \{\mathbf{x}_i\}_{i=1}^m & \exists \{C_1 \ldots, C_k\}, \left\{C'_1, \ldots, C'_k\right\} \\ \textit{For which} & G\left(C_1, \ldots, C_k\right) = G\left(C'_1, \ldots, C'_k\right) \\ \textit{It holds that} & \exists j \in [k] \;\; \forall l \in [k] \quad C_j \neq C'_l \end{array}$$

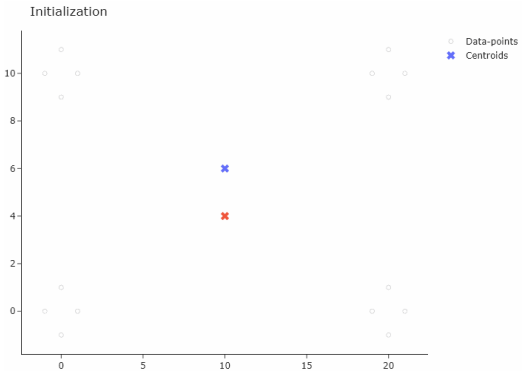


Figure 7.7: Suboptimal Solution: *Data-points and initial centroids leading to a suboptimal solution.*

and that they achieve an objective value lower (or equal) to any other partitioning of the dataset. An example for such dataset and initialization of centroids is seen in Figure 7.7. In this case both centroids are initialized in the center of all data points.

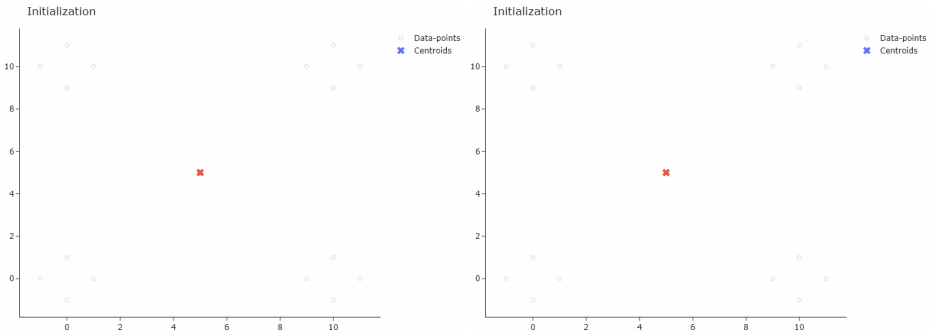


Figure 7.8: Multiple Optimal Solutions: *Data-points and initial centroids leading to two different optimal solutions*

7.2.1.2 Selection of k

As in different algorithms previously seen, in K-Means too we are required to provide a value for the hyper-parameter k. As we do not know how many clusters we really have in the dataset this is not a simple task. Notice that as long as we have more data-points than clusters, for any optimal solution with k clusters, we can achieve a solution with a lower objective for $k+1$. As such, simply running over different values of k and selecting the minimal value isn't a viable solution. Instead, we will apply a similar technique to the one used in PCA. We will plot the to objective achieved for different values of k and select the value after which the improvement is less drastic. In Figure 7.9 we apply this strategy over the dataset seen in Figure **??**. As seen in the elbow-plot once we reach $k = 4$, the incremental improvement in score is much lower. Since the data was generated from 4 different Gaussians, this approach managed to find the correct value.

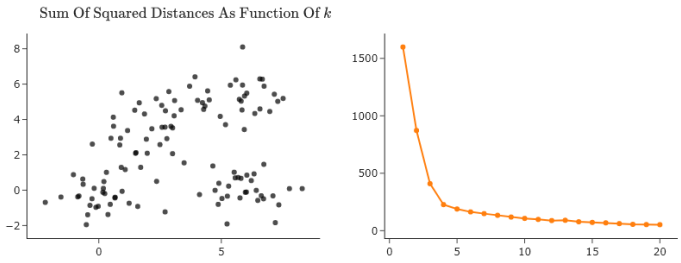
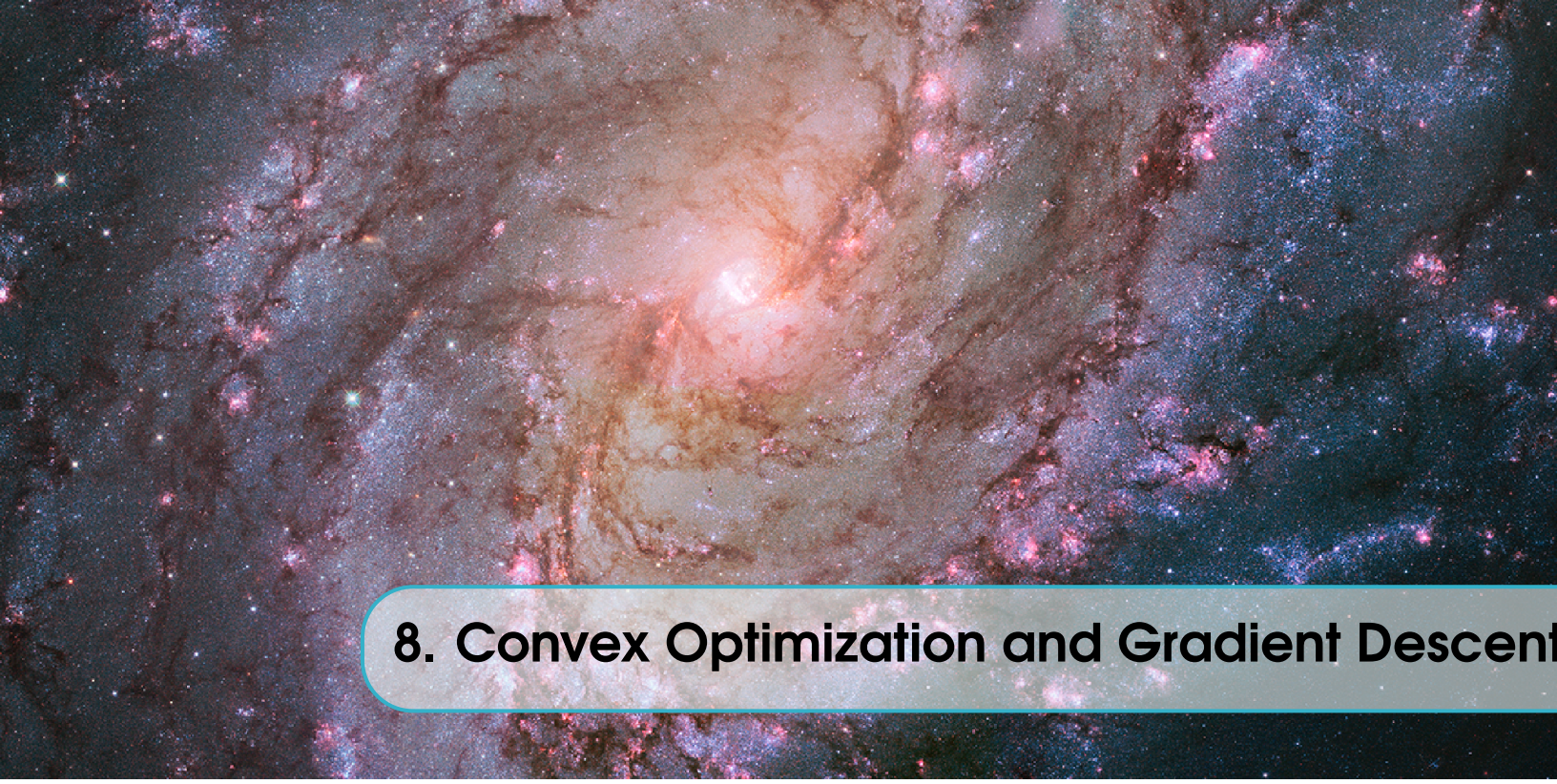


Figure 7.9: Selecting k hyper-parameter in K-Means



8. Convex Optimization and Gradient Descent

9. Online- and Reinforcement Learning

10. Deep Learning