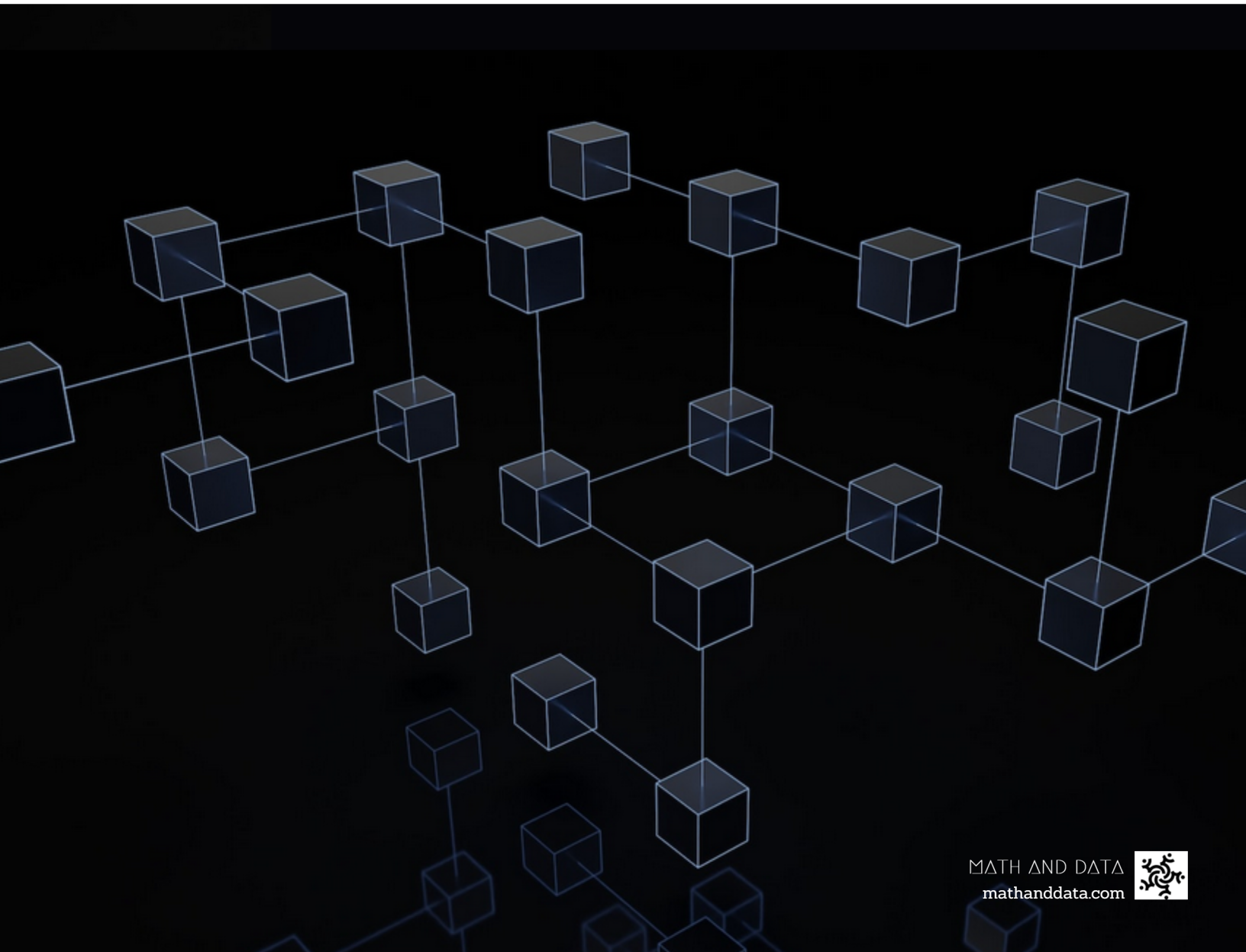
No Bullshit Math for Data Science

With Python Implementations

Your Comprehensive, Short and Sharp Guide to the Mathematics of Data Science

Amir Kiani



MATH AND DATA
mathanddata.com

Contents

Preface

About the Author and the Book

Data Science is an interdisciplinary, growing field that uses scientific methods, processes, and algorithms to extract or extrapolate knowledge, patterns, and insights from various kinds of structured, semi-structured, and unstructured data for various real-life and theoretical purposes.

To become a successful data scientist, one must have a solid foundation in mathematics, programming skills, and domain expertise. A data scientist should be able to work with large datasets, manipulate and analyze data using statistical techniques, and communicate the findings effectively to stakeholders. Furthermore, a data scientist should be able to design and develop predictive models using machine learning algorithms to solve real-world problems.

I am a data scientist, with a BSc degree in Mathematics, an MA, and a PhD degree in Philosophy, specializing in ontologies, the philosophy of AI, and data ethics. As someone who has been engaging with math, in one form or another, for over a decade, I find it disappointing that there aren't many books that teach the mathematics underneath Data Science in a comprehensive, short and sharp manner. This book aims to do exactly that.

Really, the need for writing this book is best captured by its title! There is a myriad of books on data science, machine learning and their underlying mathematics, but they're each around 350 pages long, with a high price, and only touch upon a few topics, mainly written in code, with much less actual teaching of mathematics, or doing so at advanced levels.

The book has two parts. Part I explores the core mathematics at use in data science. Part II builds upon the knowledge gained in the first part and explores the underlying mathematics of some of the well-known machine learning algorithms and models; it also provides sample codes to implement each of the algorithms in the programming language Python. The codes aren't meant to teach the reader coding from scratch; they're simply complementary material to the purely mathematical discussions in the book. While most lines of codes are explained, and you could simply copy and paste the codes into your Python environment and get similar results. Truly understanding the codes would require some basic understanding of Python.

My goal has been to keep a reasonable balance between the level of math and data science material in a way that's suitable for most practitioners and enthusiasts, particularly people who are new to the field. Given the page limits, the width of the topics covered—ranging from Statistics and Probability Theory to Linear Algebra, all the way to Calculus—and the depth each of these topics can get into (after all, people get their PhDs in sub-sub-sub-fields of each of these), one can only cover so much and needs to draw a line somewhere.

I hope you enjoy reading this book as much as I enjoyed writing it!

Make sure to check out and follow my personal blog, https://mathanddata.com, where I keep posting relevant content, including the entire content of this book, with updates implemented over time. You can simply scan this QR code to get you to my webpage:

I also run a parallel blog on the platform Medium, https://philanddata.medium.com, where I put on my professional philosopher's hat and write on the philosophical foundations and dimensions of data science, such as machine minds, artificial consciousness, data ethics, and privacy. You can simply scan this QR code to get you to my Medium blog:

Also, follow me on LinkedIn (www.linkedin.com/in/amirhossein-kiani-322a8451), where I post other relevant content, recruit people for my projects, and more. The following QR code leads you to my LinkedIn page:

How to Contribute to this Book

As I mentioned above, the content of this book will be entirely published, in HTML format, on my blog, https://mathanddata.com. This not only makes the book freely accessible to everyone, anywhere the world, but also allows for flexibility in revising, enhancing and augmenting the content of this book at any time and time.

This is one way where you can contribute: you can simply engage with the blog posts, comment on anything, ask for clarification, or propose changes in the content. You can also send me emails at info@mathanddata.com and do any of the above. In both cases, any adjustments to the content that gets to be published on the blog will mention your contribution, plus you get to plug anything there about yourself, such as your GitHub repository or LinkedIn page.

Final Words and Disclaimers

This book has come into existence through hard work, over several months, and countless hours put in. Please do not scan, download or distribute copies of the book without the consent of the author.

Also, please be aware that the contents of this book are intended for educational and entertainment purposes only. The author has taken great care to ensure that the information presented is accurate, up-to-date, and reliable, but no guarantees or warranties are made.

While most of the plots and visualizations are compiled in Python Jupyter Notebooks, when a piece of graphics is used from an external source, the source has been cited. The book has been compiled mainly using LaTeX.

By reading this document, the reader agrees that the author shall not be held responsible for any direct or indirect losses resulting from the use of the information contained herein, including but not limited to errors, omissions, or inaccuracies in the Python codes provided in the book.

Acknowledgements

I would like to thank Azim Jinha for his helpful and detailed feedback on the content of this book. He can be found at:

- LinkedIn: https://www.linkedin.com/in/azimjinha,
- GitHub: https://github.com/azimj

I would also like to thank another data scientist and my dear friend, Soohyun Ahn, for her support and encouragement throughout the process of writing this book. She can be found at:

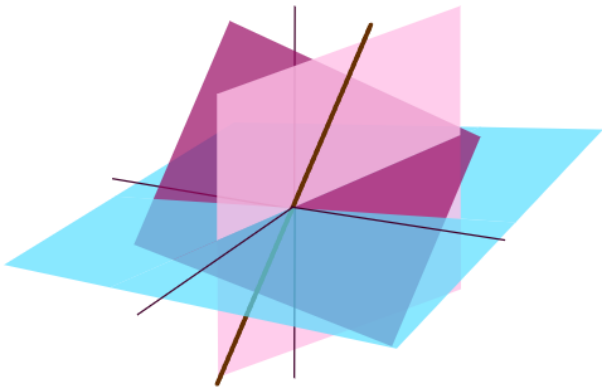
- Medium: https://medium.com/@treelunar
- GitHub: https://github.com/treelunar

Part I

The Core Mathematics

Chapter 1

Linear Algebra



1.1 Introduction

Linear algebra is a branch of mathematics that deals with the study of linear equations, matrices, vectors, and their properties.[1] Linear algebra plays a crucial role in solving many of the problems encountered in data science. Here are four reasons why linear algebra is important in data science.

Data representation: Linear algebra provides a mathematical framework for representing and manipulating data in a structured way. Data is often represented as vectors or matrices, and linear algebra operations such as matrix multiplication and inversion are used to transform and analyze this data.

Machine learning: Many machine learning algorithms are based on linear algebra, such as linear regression, principal component analysis (PCA), and singular value decomposition (SVD). Linear regression uses matrices to represent input and output data and finds the optimal coefficients for a linear function that best fits the data. PCA and SVD are used for dimensionality reduction and feature extraction, which are essential in reducing the complexity of large datasets.

Optimization: Linear algebra is also used in optimization problems, which are common in data science. Optimization problems involve finding the maximum or minimum of a function, subject to constraints. Linear algebra is used to solve these problems by transforming the objective function into a matrix form that can be optimized using standard linear algebra techniques.

In this chapter, we introduce the fundamental concepts of linear algebra, most of which will be used in the second half of the book when we attend to machine learning algorithms.

1.2 Vectors

A ***vector*** is a mathematical object that represents a quantity with both magnitude and direction. In general, a vector is represented by an ordered set of numbers, called components, that describe the direction and magnitude of the vector. (Vectors and vector spaces can be defined much more rigorously[2])

A ***real vector*** is a vector with components that belong to the set of real numbers, $\mathbb{R}$. For example, the displacement vector of a particle moving in a two-dimensional plane can be represented by a real vector with two position components, the x-component and the y-component.

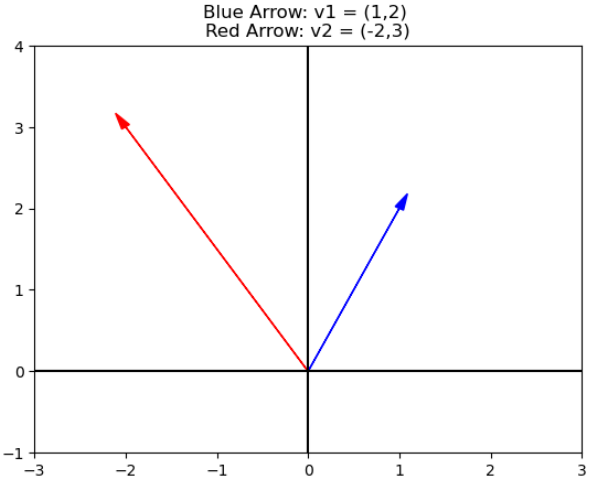
$$\vec{p} = \begin{bmatrix} x, & y \end{bmatrix} \tag{1.1}$$

It is also commonplace to show the vector above with $\vec{p} = (x, y) = \boldsymbol{p} = \mathbf{p}$. The x-component and y-component of the vector represent the displacement of the particle in two directions: horizontal (left-/right), and vertical (up/down). Sometimes vector components are also identified by index. In this case, authors may use numbered or lettered subscripts to indicate the components as in $\boldsymbol{p} = (p_1, p_2)$ or $\boldsymbol{p} = (p_x, p_y)$. (Throughout the book we use all these

notations interchangeably, so your eyes get used to all formats before you start exploring the literature!)

The ***dimension*** of a vector space is defined by the number of components used to describe a vector in that space.

For example, here's how the two-dimensional vectors $v_1 = (1, 2)$ and $v_2 = (-2, 3)$ look like in $\mathbb{R}^2$:



In what follows, suppose $\vec{v} = \begin{bmatrix} v_x, v_y \end{bmatrix}$ and $\vec{w} = \begin{bmatrix} w_x, w_y \end{bmatrix}$ are two vectors in $\mathbb{R}^2$.

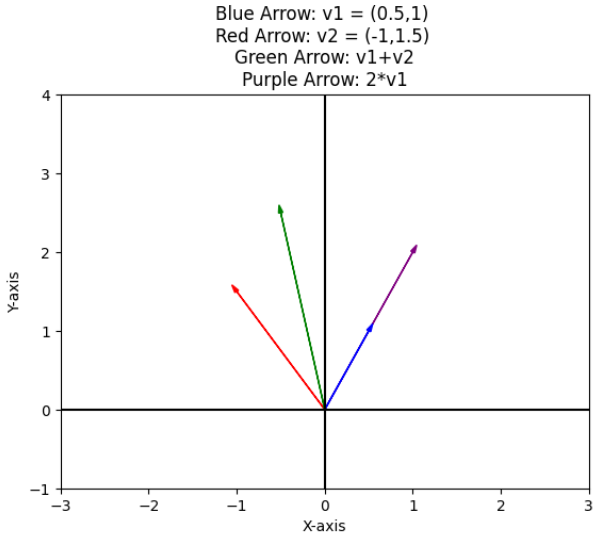
The ***addition*** of two vectors is performed by adding their corresponding components:

$$\vec{v} + \vec{w} = \begin{bmatrix} v_x, v_y \end{bmatrix} + \begin{bmatrix} w_x, w_y \end{bmatrix} = \begin{bmatrix} v_x + w_x, v_y + w_y \end{bmatrix} \quad (1.2)$$

Scaling a vector involves multiplying each of its components by a real number called scalar:[3]

$$c\vec{v} = c\begin{bmatrix} v_x, v_y \end{bmatrix} = \begin{bmatrix} cv_x, cv_y \end{bmatrix} \quad (1.3)$$

The addition and scalar product of two vectors can also be captured geometrically as long as the dimensions are low. For instance:



The ***norm*** of a vector can be thought of as a length or magnitude of a vector. An example of a norm is the *Euclidean norm* of a real vector which is defined as the square root of the sum of the squares of its components:[4]

$$|\vec{v}| = \sqrt{v_1^2 + v_2^2 + v_3^2 + \cdots + v_n^2} = \sqrt{\vec{v} \cdot \vec{v}} \quad (1.4)$$

The operations of addition and scalar multiplication on vectors have several canonical features, which are as follows:

1. *Associativity of vector addition:* This property states that the way in which we group three or more vectors to add them together does not affect the result of the addition. In mathematical notation, this property is expressed as:
$$\vec{u} + (\vec{v} + \vec{w}) = (\vec{u} + \vec{v}) + \vec{w}$$
2. *Commutativity of vector addition:* This property states that the order in which we add two vectors does not affect the result of the addition:
$$\vec{u} + \vec{v} = \vec{v} + \vec{u}$$
3. *Identity element of vector addition:* This property states that there exists a special vector, denoted by $\vec{0}$, which when added to any other vector gives that vector as the result:
$$\vec{v} + \vec{0} = \vec{v}$$
4. *Inverse elements of vector addition:* This property states that for every vector $\vec{v}$, there exists a vector $-\vec{v}$, such that when added to $\vec{v}$, the result is the identity element of vector addition:
$$\vec{v} + (-\vec{v}) = \vec{0}$$

5. *Identity element of scalar multiplication:* This property states that there exists a special scalar, denoted by 1, which when multiplied by any vector gives that vector as the result:

$$1 \cdot \vec{v} = \vec{v}$$

6. *Distributivity of scalar multiplication with respect to vector addition:* This property states that multiplying a scalar by the sum of two vectors is the same as multiplying the scalar by each vector separately and then adding the results:

$$c \cdot (\vec{u} + \vec{v}) = c \cdot \vec{u} + c \cdot \vec{v}$$

7. *Distributivity of scalar multiplication with respect to scalar addition:* This property states that multiplying a scalar by the sum of two real numbers is the same as adding the scalar multiples of each field element separately:

$$(c + d) \cdot \vec{v} = c \cdot \vec{v} + d \cdot \vec{v}$$

We close this section by introducing the notions of linear combinations and span. A ***linear combination*** of a set of vectors $\{v_1, ..., v_n\}$ is any equation of the form

$$\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_k \mathbf{v}_k \tag{1.5}$$

where each α_i is a real number – that is, $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathbb{R}$.

The ***span*** of a set of vectors is the set of all possible linear combinations of those vectors. More formally, given a set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ in a vector space V, the span of the set is defined as:

$$\text{span}(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k) = \{\alpha_1 \mathbf{v}_1 + \alpha_2 \mathbf{v}_2 + \cdots + \alpha_k \mathbf{v}_k\} \tag{1.6}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathbb{R}$.

Geometrically, the span of a set of vectors is the subspace that is formed by "stretching" and "compressing" the vectors in all possible ways.

1.2.1 Dot Product

The ***dot product*** is a mathematical operation that takes two vectors and returns a scalar value. Intuitively, the dot product of two vectors gives us a measure of how much they are aligned with each other.

The algebraic formulation of the dot product of two vectors $\vec{u}$ and $\vec{v}$ is:

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{i=1}^{n} u_i v_i \tag{1.7}$$

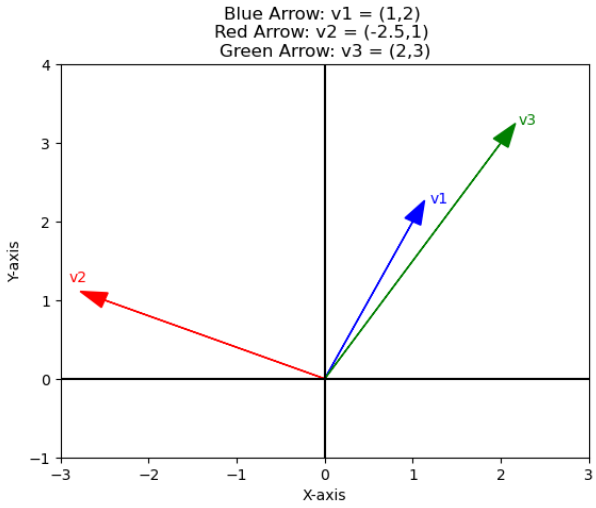
where u_i and v_i are the ith-th components of two n-dimensional the vectors $\vec{u}$ and $\vec{v}$, respectively.

If the vectors are pointing in the same direction, the dot product is positive; if they are pointing in opposite directions, the dot product is negative; and if they are orthogonal (perpendicular), the dot product is zero.

The geometric formulation of the dot product is based on the angle between the two vectors. If θ is the angle between $\vec{u}$ and $\vec{v}$, then we can write:

$$\vec{u} \cdot \vec{v} = |\vec{u}||\vec{v}| \cos(\theta) \tag{1.8}$$

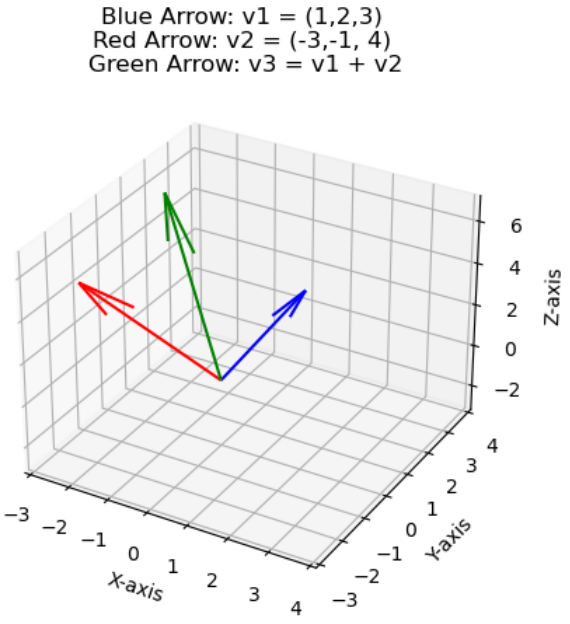
where $|\vec{u}|$ and $|\vec{v}|$ are the magnitudes (or lengths) of the vectors $\vec{u}$ and $\vec{v}$, respectively.



Dot products have the following properties:

1. *Commutativity:* $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$.
2. *Distributivity over vector addition:* $\vec{u} \cdot (\vec{v} + \vec{w}) = \vec{u} \cdot \vec{v} + \vec{u} \cdot \vec{w}$.
3. *Homogeneity:* $(c\vec{u}) \cdot \vec{v} = c(\vec{u} \cdot \vec{v}) = \vec{u} \cdot (c\vec{v})$.
4. *Non-negativity:* $\vec{u} \cdot \vec{u} \geq 0$, and $\vec{u} \cdot \vec{u} = 0$ if and only if $\vec{u}$ is the zero vector.

In the examples above we mainly used two-dimensional (2D) vectors for ease of illustration. But vectors don't have to be 2D – they can have as many dimensions as needed. Here's an example of two 3D vectors and their addition:



1.2.2 Cross Product

Intuitively, the ***cross product*** of two vectors is a vector perpendicular to both of them that gives us a measure of how much they are "rotated" with respect to each other—we will see how shortly.

Aside from physics and engineering an area where the notion of cross product is used is in machine learning and data analysis, specifically in feature engineering. Feature engineering is the process of creating new features (or variables) from existing ones to improve the performance of machine learning algorithms. The cross product can be used to create new features by taking the cross product of two existing features. For example, if we have two features representing the velocity and acceleration of an object, we can create a new feature representing the direction of the object's movement by taking the cross product of these two vectors.

The cross product is only defined for vectors in $\mathbb{R}^3$. The cross product takes two vectors in $\mathbb{R}^3$ as input and produces a third vector that is also in $\mathbb{R}^3$. This is because the cross product is defined using the properties of 3-dimensional space, such as the existence of a unique normal vector to any two non-parallel vectors.[5]

As we noted earlier, the algebraic formulation of the cross product of two vectors $\vec{u}$ and $\vec{v}$ is:

$$\vec{u} \times \vec{v} = (u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1) \quad (1.9)$$

where u_i and v_i are the i-th components of the vectors $\vec{u}$ and $\vec{v}$, respectively.

The geometric formulation of the cross product is based on the area of the parallelogram formed by the two vectors. If θ is the angle between $\vec{u}$ and $\vec{v}$, and A is the area of the parallelogram formed by them, then we can write:

$$|\vec{u} \times \vec{v}| = A = |\vec{u}||\vec{v}|\sin(\theta) \quad (1.10)$$

where $|\vec{u}|$ and $|\vec{v}|$ are the magnitudes of the vectors $\vec{u}$ and $\vec{v}$, respectively.

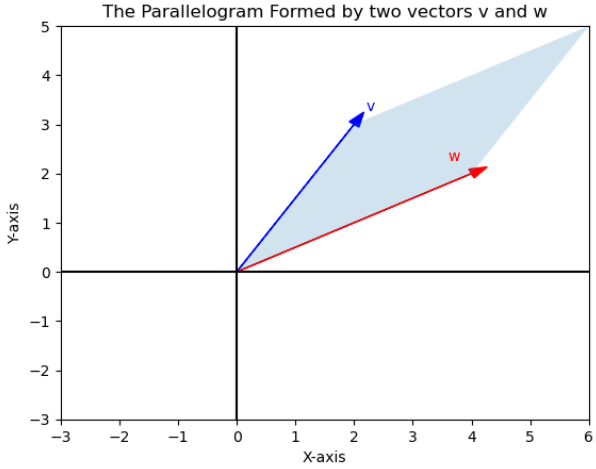
Here are the properties of cross product:

1. *Anticommutativity:* $\vec{u} \times \vec{v} = -\vec{v} \times \vec{u}$
2. *Distributivity over addition:* $\vec{u} \times (\vec{v} + \vec{w}) = \vec{u} \times \vec{v} + \vec{u} \times \vec{w}$
3. *Homogeneity:* $(c\vec{u}) \times \vec{v} = c(\vec{u} \times \vec{v}) = \vec{u} \times (c\vec{v})$

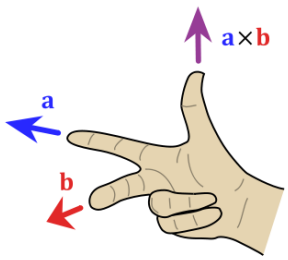
The magnitude of the cross product is equal to the product of the magnitudes of $\vec{w}$ and $\vec{v}$ times the sine of the angle between them:

$$|\vec{w} \times \vec{v}| = |\vec{u}||\vec{v}|\sin(\theta), \quad (1.11)$$

where θ is the angle between $\vec{w}$ and $\vec{v}$. The sine of θ is equal to the height of the parallelogram formed by $\vec{w}$ and $\vec{v}$ – which is also equal to the distance between the two parallel sides of the parallelogram – divided by $|u|$. Thus, the magnitude of the cross product is equal to the product of the base and height of the parallelogram, which is equal to its area.



Moreover, the direction of the cross product is perpendicular to the plane formed by the two vectors, and is determined by the right-hand rule. This means that if the two vectors $\vec{a}$ and $\vec{b}$ are oriented counterclockwise with respect to each other, then the cross product $\vec{a} \times \vec{b}$ points out of the plane towards the viewer. If the vectors are oriented clockwise with respect to each other, then the cross product points into the plane away from the viewer.[6]



Example 1.2.1. Let $\vec{u} = (1,2,3)$ and $\vec{v} = (4,5,6)$ be two real vectors. Then, the dot product of $\vec{u}$ and $\vec{v}$ is:

$$\vec{u} \cdot \vec{v} = (1,2,3) \cdot (4,5,6) = (1 \cdot 4) + (2 \cdot 5) + (3 \cdot 6) = 32$$

And the cross product of $\vec{u}$ and $\vec{v}$ is:

$$\vec{u} \times \vec{v} = (1,2,3) \times (4,5,6) = (2 \cdot 6) - \cdots - (2 \cdot 4) = (-3,6,-3)$$

To see that $\vec{u} \times \vec{v}$ is orthogonal to both u and v, you can verify that $u.(\vec{u} \times \vec{v}) = v.(\vec{u} \times \vec{v}) = 0$

1.2.3 Linear Independence

A set of vectors $\{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ is ***linearly independent*** if the only solution to the equation

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \cdots + a_n\mathbf{v}_n = \mathbf{0} \tag{1.12}$$

is the trivial solution ($a_i = 0,\ i = 1, \ldots, n$), where $a_1, a_2, \ldots, a_n$ are scalars and $\mathbf{0}$ is the zero vector.[7]

Example 1.2.2. Consider the following three vectors in $\mathbb{R}^3$:

$$\mathbf{v}_1 = \big(1,0,0\big), \quad \mathbf{v}_2 = \big(2,1,0\big), \quad \mathbf{v}_3 = \big(1,1,-1\big).$$

To show that this set is linearly independent, we need to show that the scalars a_1, a_2, a_3 are all zero, to satisfy

$$a_1\big(1,0,0\big) + a_2\big(2,1,0\big) + a_3\big(1,1,-1\big) = \big(0,0,0\big).$$

This leads to the set of equations:

$$\begin{cases} a_1 + 2a_2 + a_3 = 0 \\ a_2 + a_3 = 0 \\ a_3 = -1 \end{cases}$$

Solving for a_1, a_2, a_3, we obtain $a_3 = -1$ and $a_1 = a_2 = 1$. Therefore, we see that the set $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ is not linearly independent.

1.2.4 Basis and Dimension

A ***basis*** is a set of vectors that spans the entire vector space and is linearly independent. The dimension of a vector space is the number of vectors in any basis of that space. The ***dimension*** of a vector space is the number of vectors in any basis of that space.[8] We denote the dimension of a vector space V by $\dim(V)$.

It's important to note that a vector space may have many different bases, but every basis will have the same number of vectors, called the dimension of the vector space. For example the set of unit vectors $\big\{(1,0,0),(0,1,0),(0,0,1)\big\}$ constitutes a basis for the space $\mathbb{R}^3$, but so does $\big\{(-1,0,0),(0,-1,0),(0,0,-1)\big\}$.

To find a basis for a vector space, you need to follow these steps:

1. Determine the vectors that span the vector space. This set of vectors is not necessarily a basis yet.
2. Check for linear independence.
3. Reduce the spanning set to a linearly independent set. The resulting set of vectors is a basis for the vector space.
4. (Optional) Normalize the basis to have a unit length (i.e., norm 1).

Example 1.2.3. Let's find a basis for the vector space spanned by the following vectors in $\mathbb{R}^3$:

$$\big\{(1,2,3),(-1,0,1),(2,4,6)\big\}$$

1. Determine the vectors that span the vector space: We know from the definition of 'span' that any vector in the span of $\big\{(1,2,3),(-1,0,1),(2,4,6)\big\}$ can be expressed as a linear combination of these vectors.
2. Check for linear independence: We need to check if any of the given vectors can be expressed as a linear combination of the other two. Suppose we have

   $$a(1,2,3) + b(-1,0,1) + c(2,4,6) = 0$$

   We want to check if $a = b = c = 0$. If so, the vectors are independent; if not, they're not. This leads to the following set of equations:

   $$\begin{aligned} a - b + 2c &= 0 \\ 2a + 4c &= 0 \\ 3a + b + 6c &= 0 \end{aligned}$$

   From this, it can easily be seen that $b = 0$ and $a = -2c$. But this means that a and c can be any

numbers as long as $a = -2c$; e.g., let $c = 1$ and $a = -2$. So the vectors aren't linearly independent. In particular, $(2, 4, 6)$ can be re-written as a linear combination of the other two vectors:[9]

$$(2, 4, 6) = 2(1, 2, 3) + 0(-1, 0, 1)$$

3. Reduce the spanning set to a linearly independent set: We can remove the third vector, since it is a multiple of the first vector. Therefore, a basis for the vector space is given by the first two vectors:

$$\big\{(1, 2, 3), (-1, 0, 1)\big\}$$

4. (Optional) Normalize the basis: We can normalize each vector to have the unit length by dividing by its norm. The resulting normalized basis is:

$$\big\{\frac{1}{\sqrt{14}}(1, 2, 3), \frac{1}{\sqrt{2}}(-1, 0, 1)\big\}$$

Note that this normalized basis still spans the same vector space, but now each vector has unit length.

1.3 Matrices

A ***matrix*** is a rectangular array of numbers, symbols, or expressions arranged in rows and columns.

A general matrix can be represented as:

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{1.13}$$

where a_{ij} represents the entry in the ith row and jth column of the matrix $\boldsymbol{A}$. A matrix has that m rows and n columns and is said to have dimensions $m \times n$.

1.3.1 Matrix Operations

Matrices can be added element-wise if they have the same dimensions. For example, if $\boldsymbol{A}$ and $\boldsymbol{B}$ are $m \times n$ matrices, then their ***sum*** $\boldsymbol{A} + \boldsymbol{B}$ is defined by:

$$\boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \cdots & a_{1n} + b_{1n} \\ a_{21} + b_{21} & a_{22} + b_{22} & \cdots & a_{2n} + b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} + b_{m1} & a_{m2} + b_{m2} & \cdots & a_{mn} + b_{mn} \end{bmatrix} \tag{1.14}$$

The ***product*** of two matrices $\mathbf{A}$ and $\boldsymbol{B}$ is defined only if the number of columns in $\mathbf{A}$ is equal to the number of rows in $\boldsymbol{B}$. If $\mathbf{A}$ is an $m \times p$ matrix and $\boldsymbol{B}$ is a $p \times n$ matrix, then the product $\boldsymbol{AB}$ is an $m \times n$ matrix.

$$\begin{aligned} \boldsymbol{AB} &= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1p} \\ a_{21} & a_{22} & \dots & a_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mp} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p1} & b_{p2} & \dots & b_{pn} \end{pmatrix} \\ &= \begin{pmatrix} \sum_{k=1}^{p} a_{1k}b_{k1} & \sum_{k=1}^{p} a_{1k}b_{k2} & \dots & \sum_{k=1}^{p} a_{1k}b_{kn} \\ \sum_{k=1}^{p} a_{2k}b_{k1} & \sum_{k=1}^{p} a_{2k}b_{k2} & \dots & \sum_{k=1}^{p} a_{2k}b_{kn} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{p} a_{mk}b_{k1} & \sum_{k=1}^{p} a_{mk}b_{k2} & \dots & \sum_{k=1}^{p} a_{mk}b_{kn} \end{pmatrix} \end{aligned} \tag{1.15}$$

The ***transpose*** of a matrix $\boldsymbol{A}$ is a new matrix $\boldsymbol{A}^\top$ that is obtained by interchanging the rows and columns of $\boldsymbol{A}$. That is, if $\boldsymbol{A}$ has dimensions $m \times n$, then $\boldsymbol{A}^\top$ has dimensions $n \times m$, and the entry in the ith row and jth column of $\boldsymbol{A}^\top$ is the same as the entry in the jth row and ith column of $\boldsymbol{A}$. Formally, we can write:

$$(\boldsymbol{A}^\top)_{ij} = \boldsymbol{A}_{ji} \tag{1.16}$$

for all $i = 1, \dots, n$ and $j = 1, \dots, m$.

Example 1.3.1. Consider the following two matrices:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} -1 & 0 & 1 \\ 2 & -2 & 2 \\ -3 & 3 & -3 \end{pmatrix}$$

The sum of $\boldsymbol{A}$ and $\boldsymbol{B}$ is:

$$\begin{aligned} \boldsymbol{A} + \boldsymbol{B} &= \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} + \begin{pmatrix} -1 & 0 & 1 \\ 2 & -2 & 2 \\ -3 & 3 & -3 \end{pmatrix} \\ &= \begin{pmatrix} 1-1 & 2+0 & 3+1 \\ 4+2 & 5-2 & 6+2 \\ 7-3 & 8+3 & 9-3 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 2 & 4 \\ 6 & 3 & 8 \\ 4 & 11 & 6 \end{pmatrix} \end{aligned}$$

The product of $\boldsymbol{A}$ and $\boldsymbol{B}$ is:

$$\begin{aligned}\boldsymbol{AB} &= \begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}\begin{pmatrix}-1 & 0 & 1\\ 2 & -2 & 2\\ -3 & 3 & -3\end{pmatrix}\\ &= \begin{pmatrix}1(-1)+2(2)+3(-3) & \dots & 1(1)+2(2)+3(-3)\\ 4(-1)+5(2)+6(-3) & \dots & 4(1)+5(2)+6(-3)\\ 7(-1)+8(2)+9(-3) & \dots & 7(1)+8(2)+9(-3)\end{pmatrix}\\ &= \begin{pmatrix}-8 & 9 & -8\\ -14 & 12 & -6\\ -20 & 15 & 0\end{pmatrix}\end{aligned}$$

The transpose of $\boldsymbol{A}$ is:

$$\boldsymbol{A}^\top = \begin{pmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{pmatrix}$$

Here are some properties of the operations we defined above (note that $\boldsymbol{O}$ is the 'zero matrix', where all entries are 0:

1. $\boldsymbol{A}+\boldsymbol{B} = \boldsymbol{B}+\boldsymbol{A}$
2. $(\boldsymbol{A}+\boldsymbol{B})+\boldsymbol{C} = \boldsymbol{A}+(\boldsymbol{B}+\boldsymbol{C})$
3. $\boldsymbol{A}+\boldsymbol{O} = \boldsymbol{A}$
4. $\boldsymbol{A}+(-\boldsymbol{A}) = \boldsymbol{O}$
5. $c(\boldsymbol{AB}) = (c\boldsymbol{A})\boldsymbol{B} = \boldsymbol{A}(c\boldsymbol{B})$
6. $\boldsymbol{A}(\boldsymbol{BC}) = (\boldsymbol{AB})\boldsymbol{C}$
7. $\boldsymbol{A}(\boldsymbol{B}+\boldsymbol{C}) = \boldsymbol{AB}+\boldsymbol{AC}$
8. $(\boldsymbol{A}+\boldsymbol{B})\boldsymbol{C} = \boldsymbol{AC}+\boldsymbol{BC}$
9. $(\boldsymbol{A}^T)^T = \boldsymbol{A}$
10. $(\boldsymbol{A}+\boldsymbol{B})^T = \boldsymbol{A}^T+\boldsymbol{B}^T$
11. $(c\boldsymbol{A})^T = c\boldsymbol{A}^T$
12. $(\boldsymbol{AB})^T = \boldsymbol{B}^T\boldsymbol{A}^T$

1.3.2 Determinants

A matrix ***determinant*** is a scalar value that can be calculated for any square matrix. The determinant captures important information about the matrix, including whether it has an inverse and whether its rows or columns are linearly independent.

The determinant of an $n \times n$ matrix $\boldsymbol{A}$ can be calculated using the following recursive formula (for every given i):

$$\det(\boldsymbol{A}) = \sum_{j=1}^{n}(-1)^{i+j}a_{ij}\det(\boldsymbol{A}_{ij}) \tag{1.17}$$

where a_{ij} is the element of $\boldsymbol{A}$ in the i-th row and j-th column, and $\boldsymbol{A}_{ij}$ is the matrix obtained by deleting the i-th row and j-th column from $\boldsymbol{A}$. This formula is known as the *Laplace expansion* or *cofactor expansion* of the determinant.

For example, for a 2-by-2 matrix, this gives:

$$\det\begin{pmatrix}a & b\\ c & d\end{pmatrix} = ad-bc \tag{1.18}$$

and for a 3-by-3 matrix, it gives:

$$\det\begin{pmatrix}a & b & c\\ d & e & f\\ g & h & i\end{pmatrix} = a(ei-fh)-b(di-fg)+c(dh-eg) \tag{1.19}$$

These formulas can be used to calculate determinants by hand for small matrices. For larger matrices, however, it can be computationally expensive to calculate determinants using the Laplace expansion. Other methods, such as LU decomposition, can be used to calculate determinants more efficiently.

Intuitively, the determinant can be thought of as a measure of how much a matrix scales any given volume. For example, if we think of a matrix as a transformation that takes a unit cube to some other shape, the determinant tells us how much the volume of that shape has changed. If the determinant is positive, then the matrix scales volumes by some factor without changing their orientation, while a negative determinant indicates that the transformation also flips the orientation of the transformed shape.

Example 1.3.2. Consider the following 2×2 matrix:

$$A = \begin{bmatrix}2 & 1\\ 1 & 3\end{bmatrix}$$

The determinant of this matrix is:

$$\det(A) = (2\times 3)-(1\times 1) = 5$$

This means that when this matrix is used to transform a unit square, the resulting shape will have its area scaled by a factor of 5. This also means that the transformation will not change the orientation of the square, since the determinant is positive.

To see this, consider the square with the coordinates of its corners as $(0,0)$, $(0,1)$, $(1,1)$, and $(1,0)$. We can represent these corners as column vectors:

$$\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix},\quad \mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix},\quad \mathbf{p}_3 = \begin{bmatrix}1\\1\end{bmatrix},\quad \mathbf{p}_4 = \begin{bmatrix}1\\0\end{bmatrix}$$

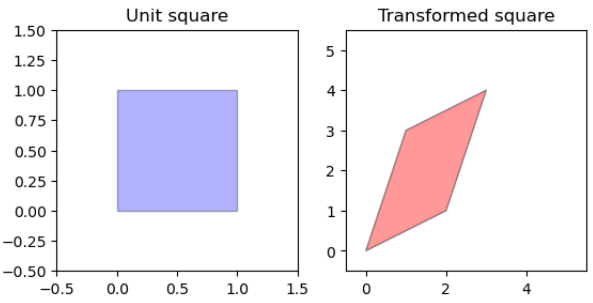
We can then stack these column vectors into a matrix:

$$P = \begin{bmatrix}0 & 0\\ 0 & 1\\ 1 & 1\\ 1 & 0\end{bmatrix}$$

This matrix represents the unit square in the xy-plane. To apply the transformation given by the matrix A, we multiply P by A:

$$AP = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 3 \\ 3 & 4 \\ 2 & 1 \end{bmatrix}$$

The resulting matrix AP represents the transformed unit square.



1.3.3 Matrix Inverse

The ***inverse*** of a square matrix of $n \times n$ dimension $\boldsymbol{A}$, denoted as $\boldsymbol{A}^{-1}$, is a matrix such that the product of $\boldsymbol{A}$ and $\boldsymbol{A}^{-1}$ is the identity matrix

$$\boldsymbol{I} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}_{n \times n}$$

In other words, if $\boldsymbol{A}$ is an $n \times n$ matrix, then $\boldsymbol{A}^{-1}$ a matrix satisfying:

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I} \tag{1.20}$$

The inverse of a matrix exists if and only if the determinant of the matrix is nonzero, i.e., $\det(\boldsymbol{A}) \neq 0$.

There are many ways to calculate the inverse of a matrix, both directly and indirectly. Below we'll discuss some of these.

1.3.3.1 Direct Method

The formula to compute the inverse of a matrix directly is:

$$\boldsymbol{A}^{-1} = \frac{1}{\det(\boldsymbol{A})} \text{adj}(\boldsymbol{A}) \tag{1.21}$$

where $\text{adj}(\boldsymbol{A})$ is the ***adjugate*** of $\boldsymbol{A}$, which is obtained by taking the transpose of the matrix of the ***cofactors*** of A. The cofactor of the element in the ith row and jth column of $\boldsymbol{A}$ is denoted by $\boldsymbol{C}_{ij}$ and is defined as:

$$\boldsymbol{C}_{ij} = (-1)^{i+j} \det(\boldsymbol{A}_{ij}) \tag{1.22}$$

where $\boldsymbol{A}_{ij}$ is the $(n-1) \times (n-1)$ matrix obtained by deleting the ith row and jth column of $\boldsymbol{A}$.

Example 1.3.3. Let's say we want to find the inverse of the matrix $\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ 3 & 4 \end{pmatrix}$. First, we compute its determinant:

$$\det(\boldsymbol{A}) = 2 \cdot 4 - 1 \cdot 3 = 5$$

Since the determinant is nonzero, we can proceed to compute the adjugate of $\boldsymbol{A}$ using the cofactor formula:

$$\boldsymbol{C}_{11} = (-1)^{1+1} \det\begin{pmatrix} 4 \end{pmatrix} = 4$$

$$\boldsymbol{C}_{21} = (-1)^{2+1} \det\begin{pmatrix} 1 \end{pmatrix} = -1$$

$$\boldsymbol{C}_{12} = (-1)^{1+2} \det\begin{pmatrix} 3 \end{pmatrix} = -3$$

$$\boldsymbol{C}_{22} = (-1)^{2+2} \det\begin{pmatrix} 2 \end{pmatrix} = 2$$

Therefore, the adjugate of $\boldsymbol{A}$ is:

$$\text{adj}(\boldsymbol{A}) = \begin{pmatrix} 4 & -3 \\ -1 & 2 \end{pmatrix}^T = \begin{pmatrix} 4 & -1 \\ -3 & 2 \end{pmatrix}$$

Finally, we can use the formula for the inverse to compute:

$$\boldsymbol{A}^{-1} = \frac{1}{\det(\boldsymbol{A})} \text{adj}(\boldsymbol{A}) = \frac{1}{5} \begin{pmatrix} 4 & -1 \\ -3 & 2 \end{pmatrix} = \begin{pmatrix} \frac{4}{5} & -\frac{1}{5} \\ -\frac{3}{5} & \frac{2}{5} \end{pmatrix}$$

Therefore, the inverse of $\boldsymbol{A}$ is $\begin{pmatrix} \frac{4}{5} & -\frac{1}{5} \\ -\frac{3}{5} & \frac{2}{5} \end{pmatrix}$.

While the formula $\boldsymbol{A}^{-1} = \frac{1}{\det(\boldsymbol{A})} \text{adj}(\boldsymbol{A})$ provides a direct way to compute the inverse of a matrix, it may not always be the most efficient or numerically stable method. In some cases, it may be faster or more accurate to use other methods.

For example, when dealing with large matrices, some methods can be more efficient than computing the determinant and adjugate directly. This calls for indirect methods of finding matrix inverses.

1.3.3.2 Indirect Methods

There are several methods to obtain matrix inverses. In this section we explain the famous Gaussian elimination method.

The Gaussian elimination method is a technique used to solve systems of linear equations and to compute the inverse of a matrix.

Here is a step-by-step explanation of how to use Gaussian elimination to calculate the inverse of a matrix:

1. Given an $n \times n$ matrix $\boldsymbol{A}$, augment it with the $n \times n$ identity matrix I to obtain the augmented matrix $[\boldsymbol{A} \,|\, \boldsymbol{I}]$.
2. Use row operations to transform the augmented matrix into the form $[\boldsymbol{I} \,|\, \boldsymbol{A}^{-1}]$, where $\boldsymbol{A}^{-1}$ is the inverse of the original matrix $\boldsymbol{A}$.

This is best explained through examples. Here's one:

Example 1.3.4. Let's say we want to compute the inverse of the following matrix using Gaussian elimination:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix}$$

1. Augment the matrix $\boldsymbol{A}$ with the identity matrix $\boldsymbol{I}$:

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 4 & 0 & 1 & 0 \\ 5 & 6 & 0 & 0 & 0 & 1 \end{array}\right]$$

2. Use row operations to transform the augmented matrix into the form $[\boldsymbol{I} \,|\, \boldsymbol{A}^{-1}]$.

   a. Subtract 5 times the first row from the third row ($R_3 = R_3 - 5R_1$) to obtain:

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 4 & 0 & 1 & 0 \\ 0 & -4 & -15 & -5 & 0 & 1 \end{array}\right]$$

   b. Add 4 times the second row to the third row ($R_3 = 4R_2 + R_3$) to obtain:

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 3 & 1 & 0 & 0 \\ 0 & 1 & 4 & 0 & 1 & 0 \\ 0 & 0 & 1 & -5 & 4 & 1 \end{array}\right]$$

   c. Subtract 3 times the third row from the first row ($R_1 = R_1 - 3R_3$) to obtain:

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & 16 & -12 & -3 \\ 0 & 1 & 4 & 0 & 1 & 0 \\ 0 & 0 & 1 & -5 & 4 & 1 \end{array}\right]$$

   d. Subtract 2 times the third row from the second row ($R_2 = R_2 - 4R_3$) to obtain:

$$\left[\begin{array}{ccc|ccc} 1 & 2 & 0 & 16 & -12 & -3 \\ 0 & 1 & 0 & 20 & -15 & -4 \\ 0 & 0 & 1 & -5 & 4 & 1 \end{array}\right]$$

   e. Subtract 2 times the second row from the first row ($R_1 = R_1 - 2R_2$) to obtain:

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & -24 & 18 & 5 \\ 0 & 1 & 0 & 20 & -15 & -4 \\ 0 & 0 & 1 & -5 & 4 & 1 \end{array}\right]$$

Therefore: $\boldsymbol{A}^{-1} = \begin{pmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{pmatrix}$.

We can verify the result by multiplying $\boldsymbol{A}$ and its inverse:

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 4 \\ 5 & 6 & 0 \end{pmatrix} \begin{pmatrix} -24 & 18 & 5 \\ 20 & -15 & -4 \\ -5 & 4 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

as expected.

1.3.4 Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors are a pair of important concepts in linear algebra that arise in a variety of applications, including physics, engineering, and computer science.

The set of eigenvectors of a matrix is a special set of input vectors for which the action of the matrix is described as a simple scaling. An ***eigenvector*** of a matrix is a non-zero vector that, when multiplied by the matrix, is scaled by a scalar factor known as its ***eigenvalue***.

Mathematically, given a square matrix $\boldsymbol{A}$, an eigenvector x and an eigenvalue λ satisfy the following equation:

$$\boldsymbol{A}x = \lambda x \tag{1.23}$$

where x is a non-zero vector and λ is a scalar. In other words, when we multiply $\boldsymbol{A}$ by the eigenvector x, we get a scaled version of x. The eigenvalue λ represents the scaling factor.

To calculate the eigenvectors and eigenvalues of a matrix $\boldsymbol{A}$, we need to solve the characteristic equation:

$$\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0 \tag{1.24}$$

where $\boldsymbol{I}$ is the identity matrix of the same size as $\boldsymbol{A}$. The roots of this equation are the eigenvalues of $\boldsymbol{A}$.

Once we have the eigenvalues, we can find the eigenvectors by solving the system of linear equations:

$$(\boldsymbol{A} - \lambda \boldsymbol{I})x = 0 \tag{1.25}$$

for each eigenvalue λ. The solutions to this equation form a vector space known as the ***eigenspace*** corresponding to λ. The eigenvectors are then chosen to be a basis for each eigenspace.

Intuitively, the eigenvector "points in the same direction" as its image under the matrix transformation, but its length is stretched or compressed by the eigenvalue. In other words, when a matrix $\boldsymbol{A}$ is multiplied by an eigenvector x, the resulting vector is in the same direction as x, but its magnitude is changed. This change in magnitude is determined by the corresponding eigenvalue λ.

It is important to note that not all matrices have eigenvectors or eigenvalues. A matrix that has eigenvalues and eigenvectors is known as ***diagonalizable***. A necessary and sufficient condition for a matrix to be diagonalizable is that it has n linearly independent eigenvectors, where n is the size of the matrix.

Eigenvalues and eigenvectors have important applications in various fields, such as computer graphics, signal processing, quantum mechanics, and data science. They are used to understand the behavior of linear transformations and to simplify complex calculations involving matrices. Later we will get back to these notions when discussing the machine learning algorithm known as Principal Component Analysis.

Example 1.3.5. Suppose we have the following matrix:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$$

To find the eigenvalues and eigenvectors of $\boldsymbol{A}$, we start by solving the characteristic equation:

$$\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0$$

where I is the identity matrix and λ is the eigenvalue we want to find. For our matrix $\boldsymbol{A}$, this becomes:

$$\begin{aligned} \det(\boldsymbol{A} - \lambda \boldsymbol{I}) &= \begin{vmatrix} 1-\lambda & 2 \\ 2 & 1-\lambda \end{vmatrix} = (1-\lambda)^2 - 4 \\ &= \lambda^2 - 2\lambda - 3 = (\lambda - 3)(\lambda + 1) \end{aligned}$$

So the eigenvalues of $\boldsymbol{A}$ are $\lambda_1 = 3$ and $\lambda_2 = -1$.

Next, we need to find the eigenvectors corresponding to each eigenvalue. For each eigenvalue λ, we solve the equation:

$$(\boldsymbol{A} - \lambda \boldsymbol{I})\mathbf{v} = 0$$

where $\mathbf{v}$ is the eigenvector corresponding to λ. For $\lambda_1 = 3$, this becomes:

$$(\boldsymbol{A} - 3\boldsymbol{I})\mathbf{v} = \begin{pmatrix} -2 & 2 \\ 2 & -2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Solving this system of equations, we get $x = y$, so any vector of the form $\big(a, a\big)$ is an eigenvector corresponding to λ_1. Similarly, for $\lambda_2 = -1$, we get:

$$(\boldsymbol{A} + \boldsymbol{I})\mathbf{v} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Solving this system of equations, we get $x = -y$, so any vector of the form $\big(a, -a\big)$ is an eigenvector corresponding to λ_2.

In this example, the matrix $\boldsymbol{A}$ has two eigenvalues ($\lambda_1 = 3$ and $\lambda_2 = -1$) and two corresponding eigenvectors (for $a = 1$, $\mathbf{v}_1 = \big(1, 1\big)$ and $\mathbf{v}_2 = \big(1, -1\big)$). This means that when we apply the matrix transformation represented by $\boldsymbol{A}$ to these vectors, they simply get scaled by a factor of 3 and -1, respectively. Geometrically, we can think of this as stretching or compressing the vectors along certain directions.

For example, if we apply $\boldsymbol{A}$ to $\mathbf{v}_1$, we get:

$$\boldsymbol{A}\mathbf{v}_1 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} = \lambda_1 \mathbf{v}_1$$

Similarly, if we apply $\boldsymbol{A}$ to $\mathbf{v}_2$, we get:

$$\boldsymbol{A}\mathbf{v}_2 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \lambda_2 \mathbf{v}_2$$

The process of decomposing a matrix into its eigenvalues and eigenvectors is called ***eigendecomposition***. Eigendecomposition can provide valuable insights into the properties of a matrix, such as its symmetry, invertibility, and positive definiteness.

Google's original PageRank algorithm uses eigendecomposition to rank webpages by importance. The algorithm models the web as a directed graph, where each webpage is a node and each hyperlink is an edge. The matrix of web hyperlinks is called the hyperlink matrix.

The PageRank algorithm formalizes as an eigenvector calculation on the hyperlink matrix, where the eigenvector corresponding to the largest eigenvalue is considered to be the PageRank vector. This vector represents the importance of each webpage in the web graph, with more important pages having higher PageRank values.[10]

1.3.5 Row Space, Column Space, and Rank

The row space and column space of a matrix are important subspaces associated with the matrix, while the rank is a measure of the "size" of these subspaces.

The ***row space*** of an m-by-n matrix $\boldsymbol{A}$ is the set of all linear combinations of the rows of $\boldsymbol{A}$:

$$\text{Row}(\boldsymbol{A}) = \text{span}(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_m) \tag{1.26}$$

where $\mathbf{r}_i$ is the ith row of $\boldsymbol{A}$.

The row space can be thought of as the space of all possible linear combinations of the rows of $\boldsymbol{A}$. It is a subspace of $\mathbb{R}^n$ and has dimension at most m, the number of rows in $\boldsymbol{A}$.

The ***column space*** of a matrix $\boldsymbol{A}$ is the set of all linear combinations of the columns of $A\boldsymbol{A}$:

$$\text{Col}(\boldsymbol{A}) = \text{span}(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n) \tag{1.27}$$

where $\mathbf{c}_i$ is the i-th column of $\boldsymbol{A}$.

Finally, the ***rank*** of a matrix $\boldsymbol{A}$, denoted by $\text{rank}(\boldsymbol{A})$, is the dimension of its row space (or equivalently, the dimension of its column space). The rank of a matrix can be thought of as a measure of the "size" of the row and column spaces of the matrix.

Note that the row space and column space of A are not equal, since the matrix is not square. If the matrix were square, then the row space and column space would be equal by the following theorem:

Theorem 1.3.1 (Rank-Nullity Theorem)**.**

$$\text{rank}(\boldsymbol{A}) + \text{nullity}(\boldsymbol{A}) = n \tag{1.28}$$

where n is the number of columns in $\boldsymbol{A}$.

We conclude this chapter by laying down a canonical result that connects most of the concepts we've learned so far:

Theorem 1.3.2. For an $n \times n$ matrix $\boldsymbol{A}$, the following statements are equivalent:

1. $\boldsymbol{A}$ is invertible
2. The rank of the matrix is n
3. The row space of $\boldsymbol{A}$ is $\mathbb{R}^n$
4. The column space of $\boldsymbol{A}$ is $\mathbb{R}^n$
5. The determinant of $\boldsymbol{A}$ is nonzero $\det(\boldsymbol{A}) \neq 0$

Notes

[1]The image above is adapted from https://en.wikipedia.org/wiki/Linear_algebra

[2]Rigorously put, a ***vector space*** $\mathbf{V}$ over a field $\mathbb{F}$ is a set of objects, called vectors, along with two operations: (i) *Vector addition:* For any two vectors $\mathbf{u}$ and $\mathbf{v}$ in V, there is a vector $\mathbf{u}+\mathbf{v}$ in V, called the sum of $\mathbf{u}$ and $\mathbf{v}$; (ii) *scalar multiplication:* For any scalar a in $\mathbb{F}$ and any vector $\mathbf{u}$ in V, there is a vector $a\mathbf{u}$ in V, called the scalar multiple of $\mathbf{u}$ by a. These operations satisfy the following axioms. A ***subspace*** of $\mathbf{V}$ is a subset of $\mathbf{V}$ that is itself a vector space over $\mathbb{F}$ with respect to the operations of vector addition and scalar multiplication inherited from $\mathbf{V}$. Note that the first condition above ensures that $\mathbf{W}$ is non-empty since it contains at least the zero vector. Also note that the subspace $\mathbf{W}$ of $\mathbf{V}$ inherits all the other axioms of a vector space from $\mathbf{V}$, such as associativity, commutativity, distributivity, and so on. Examples of subspaces of a vector space $\mathbf{V}$ include the trivial subspace $\mathbf{0}$, the entire space $\mathbf{V}$ itself, and any linearly independent subset of $\mathbf{V}$. A ***field*** is a set $\mathbb{F}$ with two binary operations, usually denoted as addition $(+)$ and multiplication $(\cdot)$—sometimes denoted by $(\mathbb{F}, +, \cdot)$—that satisfy the following axioms: (i) $\forall a, b \in \mathbb{F},\ a+b \in \mathbb{F}$ and $a \cdot b \in \mathbb{F}$ (Closure under addition and multiplication); (ii) $\forall a, b, c \in \mathbb{F},\ (a+b)+c = a+(b+c)$ and $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ (Associativity of addition and multiplication); (iii) $\exists 0 \in \mathbb{F},\ \forall a \in \mathbb{F},\ a+0 = a$ and $\exists 1 \in \mathbb{F},\ \forall a \in \mathbb{F},\ a \cdot 1 = a$ (Existence of additive and multiplicative identities); (iv) $\forall a \in \mathbb{F},\ \exists(-a) \in \mathbb{F},\ a+(-a) = 0$ (Existence of additive inverses); (v) $\forall a \in \mathbb{F} \setminus 0,\ \exists a^{-1} \in \mathbb{F},\ a \cdot a^{-1} = 1$ (Existence of multiplicative inverses); (vi) $\forall a, b \in \mathbb{F},\ a+b = b+a$ and $a \cdot b = b \cdot a$ (Commutativity of addition and multiplication). Examples of fields include the real numbers $\mathbb{R}$, and the complex numbers $\mathbb{C}$.

[3]The subtraction of two vectors, $\vec{v} - \vec{w}$ is defined as $\vec{v} + (-1.\vec{w})$.

[4]This is just one type of norm. Other norms do find their use in machine learning or other branches of sciences and mathematics. The *Euclidean norm* is just one of many functions $\mathbb{R}^n \to \mathbb{R}$ that have certain properties and that a vector space with a norm is thought of as a 'normed vector space'.

[5]In contrast, in $\mathbb{R}^2$, there is no unique vector that is orthogonal to any two non-parallel vectors, which means that the cross product cannot be defined in this space. However, there is a related operation called the "wedge product" or "exterior product" that is defined for vectors in any dimension, although it has different properties and applications compared to the cross product.

[6]The following image is taken from https://en.wikipedia.org/wiki/Right-hand_rule

[7]The above equation can also be written in matrix form as $\boldsymbol{A}\mathbf{x} = \mathbf{0}$, where $\boldsymbol{A}$ is an $n \times m$ matrix whose columns are the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, $\mathbf{x}$ is an $m \times 1$ column vector of scalars, and $\mathbf{0}$ is an $n \times 1$ column vector of zeros. The set of vectors is linearly independent if and only if the only solution to this equation is $\mathbf{x} = \mathbf{0}$. We will learn about matrices later in the chapter.

[8]This number is well-defined because it can be proved that any two bases for the same vector space have the same number of vectors.

[9]Here's an alternative way to check linear independence using matrices (see the next section for matrices and row reduction): writing the vectors as columns of a matrix and row reducing, we obtain: $\begin{pmatrix} 1 & -1 & 2 \\ 2 & 0 & 4 \\ 3 & 1 & 6 \end{pmatrix} \to \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$. Since the third column is not a pivot column, we can express it as a linear combination of the first two columns. Therefore, the given vectors are not linearly independent.

[10]Check out this article for more on this: https://en.wikipedia.org/wiki/PageRank

1.4 Further Reading

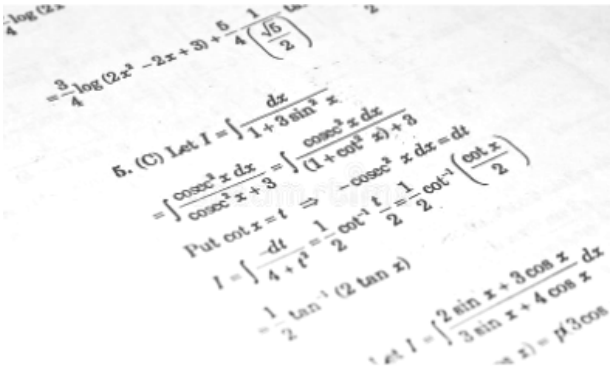
Here are some references that discuss vectors, matrices, their operations, and other concepts in linear algebra, at different levels of depth and rigor:

1. *Linear Algebra Done Right*, by Sheldon Axler [4]. This book is a popular choice for beginners in linear algebra. It provides an intuitive approach to linear algebra concepts and proofs and covers topics such as vector spaces, linear transformations, eigenvalues and eigenvectors, and inner product spaces.
2. *Linear Algebra and Its Applications*, by Gilbert Strang [40]. This book is widely used in undergraduate linear algebra courses. It covers topics such as matrix algebra, determinants, vector spaces, eigenvalues and eigenvectors, and linear transformations. The book includes many examples and exercises to reinforce the concepts.
3. *Matrix Analysis and Applied Linear Algebra*, by Carl D. Meyer [24]. This book is a comprehensive introduction to linear algebra and its applications. It covers topics such as matrix algebra, eigenvalues and eigenvectors, singular value decomposition, and linear least squares. The book also includes numerous applications to fields such as signal processing, statistics, and numerical analysis.
4. *Advanced Linear Algebra*, by Steven Roman [31]. This book is aimed at students who have already studied linear algebra and want to deepen their understanding of the subject. It covers topics such as abstract vector spaces, linear transformations, matrices and determinants, inner product spaces, and spectral theory. The book includes many exercises and examples to reinforce the concepts.
5. *Numerical Linear Algebra*, by Lloyd N. Trefethen and David Bau III [42]. This book focuses on the numerical aspects of linear algebra, and covers topics such as matrix factorizations, iterative methods, eigenvalue computations, and singular value decomposition. The book is aimed at advanced undergraduate and graduate students in mathematics, engineering, and the sciences.

Chapter 2

Calculus

2.1 Introduction



Calculus is a branch of mathematics that deals with the study of rates of change and continuous processes. It includes two main branches: differential calculus and integral calculus. Differential calculus is concerned with the instantaneous rate of change of a function, while integral calculus deals with the accumulation of infinitesimal changes over an interval.

Calculus is an essential tool in many fields, including physics, engineering, economics, and data science. In data science, calculus is used extensively to develop algorithms for optimization, prediction, and machine learning. For example, gradient descent, a popular optimization algorithm used in machine learning, relies heavily on calculus concepts such as partial derivatives and the chain rule (we will explore the applications of calculus in the second half of the book). Similarly, differential equations, which are a fundamental part of calculus, are used in modeling and analyzing complex systems in data science.

2.2 Functions

The foundation of calculus is the notion of a function. A function is a mathematical rule that assigns a unique output for every input. In other words, a function is a relationship between two sets, where each input has exactly one output.

More formally, a ***function*** f is a set of ordered pairs (x, y) such that for each x in the 'domain' of f, there is exactly one y in the 'co-domain' of f such that $(x, y) \in f$. We write this as:

$$f : A \to B \tag{2.1}$$

where A is the domain of f and B is the c–domain of f.

The ***domain*** of a function f, denoted by $\mathrm{Dom}(f)$, is the set of all possible inputs for which the function is defined, and the ***co-domain*** of f, denoted by $\mathrm{Cod}(f)$, is the set where the elements of the domain are mapped to. The ***range*** of a function f, denoted by $\mathrm{Ran}(f)$, is the set of all possible outputs that the function can produce—that is, $\mathrm{Ran}(f) = \{f(x) \,|\, x \in \mathrm{Dom}(f)\}$.

For example, the following are two functions, each taking any given element x in the domain to an element $x + 1$ in the range:

$$\begin{cases} f : \mathbb{R} \to \mathbb{R}, \ x \mapsto x + 1 \\ g : \mathbb{N} \to \mathbb{N}, \ x \mapsto x + 1 \end{cases}$$

Note that while these functions have the same mapping mechanism, they're different, as their domains and co-domains are different. Assuming the domain and co-domain of a function $f : A \to B$ are known to the context, f is sometimes by convention represented by its mapping component, as $f(x)$. For example, the functions above each can be represented as follows: $f(x) = x + 1$ and $g(x) = x + 1$.

An ***inverse*** of a function $f : A \to B$ is a function with the domain and co-domain of f swapped, that "undoes" the original function, and is usually denoted by $f^{-1} : B \to A$. That is, if we apply the original function followed by the inverse function, we

get back the input we started with; and vice versa. Formally put: $f^{-1}(f(x)) = x$, and $f(f^{-1}(x)) = x$ for all $x \in A$.

Functions that operate on sets of numbers (e.g., $\mathbb{R}$ or $\mathbb{N}$) can be plotted in the space; this usually helps develop insights about the behavior of the function.

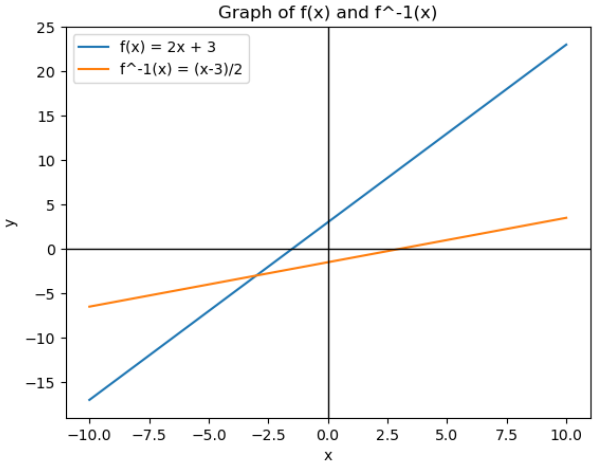
Example 2.2.1. Consider the function $f(x) = 2x+3$ with domain and co-domain $\mathbb{R}$. To find its inverse function, we first write $y = 2x + 3$, and solve for x in terms of y:

$$y = 2x + 3 \qquad\qquad x = \frac{y-3}{2}$$

So the inverse function of $f(x) = 2x + 3$ is $f^{-1}(x) = \frac{x-3}{2}$. We can check that these functions are indeed inverses by verifying that $f(f^{-1}(x)) = x$ and $f^{-1}(f(x)) = x$:

$$f(f^{-1}(x)) = f(\frac{x-3}{2}) = 2(\frac{x-3}{2}) + 3 = x$$
$$f^{-1}(f(x)) = f^{-1}(2x+3) = \frac{2x+3-3}{2} = x$$

Here's the plot of these two functions in the Cartesian plane:



Here's a list of some common functions and their inverses (try them on your own!):

$$\begin{cases} f(x) = x + 2 \Leftrightarrow f^{-1}(x) = x - 2 \\ f(x) = 2x \Leftrightarrow f^{-1}(x) = \frac{1}{2}x \\ f(x) = x^2 \Leftrightarrow f^{-1}(x) = \sqrt{x} \\ f(x) = w^x \Leftrightarrow f^{-1}(x) = \log_w(x) \\ f(x) = 3x + 5 \Leftrightarrow f^{-1}(x) = \frac{x-5}{3} \\ f(x) = a^x \Leftrightarrow f^{-1}(x) = \log_a(x) \\ f(x) = e^x \Leftrightarrow f^{-1}(x) = \ln(x) := \log_e(x) \\ f(x) = \sin(x) \Leftrightarrow f^{-1}(x) = \sin^{-1}(x) \\ f(x) = \cos(x) \Leftrightarrow f^{-1}(x) = \cos^{-1}(x) \end{cases}$$

2.3 Limits

A ***limit*** is a mathematical concept that describes the behavior of a function as the input approaches a particular value—a number or infinity. It provides a way of understanding what happens to the function as the input gets very close to the specified value. Intuitively, we can think of this as the "limiting value" of the function as the input gets arbitrarily close to the value we are interested in.

We say that the limit of a function $f(x)$ as x 'approaches' a number a is L if we can make $f(x)$ arbitrarily close to L by making x sufficiently close to, but not equal to, a.

There are three possible outcomes for the limit:

1. The limit exists and is equal to a finite value, which means that the function approaches a single value as the input approaches a.
2. The limit does not exist, which means that the function does not approach a single value as the input approaches a.
3. The limit exists, but is infinite, which means that the function grows without bound as the input approaches a.

Let's see an example (don't worry; you're owed an explanation of what's going on in what follows!)

Example 2.3.1. Let's consider the function $f(x) = \frac{x^2-1}{x-1}$. If we try to evaluate f directly at $x = 1$, we get an undefined expression (since division by zero is not allowed). However, we can investigate the limit of $f(x)$ as x approaches 1:

$$\lim_{x\to 1} \frac{x^2-1}{x-1} = \lim_{x\to 1} \frac{(x-1)(x+1)}{x-1} \qquad = \lim_{x\to 1}(x+1) \ = 2$$

So the limit of $f(x)$ as x approaches 1 exists and is equal to 2.

2.3.1 Limits to a Number

The rigorous definition of a limit at a number is a mathematical statement that describes the behavior of a function as its input approaches a specific value – the kind of behavior we just saw.

Formally, we say that the limit of a function $f(x)$ as x approaches a is L, if for a chosen arbitrarily small δ there exists an ϵ such that if $|x - a| < \delta$ then $|f(x) - L| < \epsilon$.

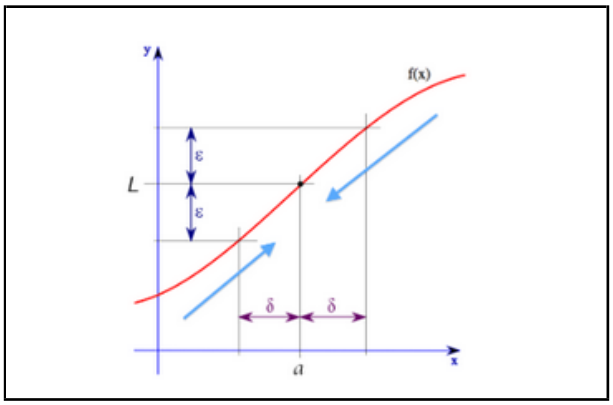
In symbols:

$$\lim_{x\to a} f(x) = L \tag{2.2}$$

if and only if

$$\forall \epsilon > 0 \, \exists \delta > 0 \, (0 < |x - a| < \delta \rightarrow |f(x) - L| < \epsilon), \quad (2.3)$$

where $\forall$, called the universal quantifier, stands for 'for all' and $\exists$, called the existential quantifier, stands for 'there is a'.[11]



This definition states that no matter how small we choose ϵ to be, we can always find a small enough neighborhood of a (i.e., the interval $(a-\delta, a+\delta)$) such that the function $f(x)$ is always within ϵ units of the limit L.

The rigorous definition of limit at a number is important in calculus because it allows us to make precise statements about the behavior of functions, such as their continuity and differentiability at specific points.

Example 2.3.2. Suppose we want to prove that the function $f(x) = x^2$ has a limit of 4 as x approaches 2. We need to show that for every $\epsilon > 0$, there exists a $\delta > 0$ such that $|f(x) - 4| = |x^2 - 4| < \epsilon$ whenever $0 < |x-2| < \delta$. To do this, we can choose $\delta = \sqrt{\epsilon + 4} - 2$, since if $0 < |x - 2| < \delta$, then:

$$|x^2 - 4| = |x-2||x+2| < \delta|x+2| < (\sqrt{\epsilon+4}-2)(\sqrt{\epsilon+4}+2) = \epsilon$$

Note that from $0 < |x - 2| < \delta$ it follows that $\delta < x + 2 < 4 + \delta = \sqrt{\epsilon + 4} + 2$.

2.3.1.1 Left and Right Limits

Left and right limits are used when the function approaches a limit from the left- or right-hand side of a point.

The ***left-hand limit*** of a function $f(x)$ as x approaches a, denoted by $\lim_{x \to a^-} f(x)$, represents the behavior of the function as x approaches a from the left-hand side. It considers the values of $f(x)$ when x is slightly less than a, but not equal to a.

The ***right-hand limit*** of a function $f(x)$ as x approaches a, denoted by $\lim_{x \to a^+} f(x)$, represents the behavior of the function as x approaches a from the right-hand side. It considers the values of $f(x)$ when x is slightly greater than a, but not equal to a.

The rigorous definitions of left and right limits are similar to the definition of a limit, but with some modifications.

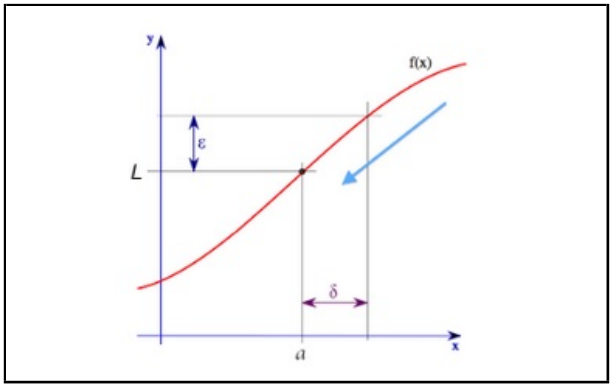
We say that the right-hand limit of $f(x)$ as x approaches a is L if for every positive number ϵ, there exists a corresponding positive number δ such that for all x satisfying $a < x < a + \delta$, the distance between $f(x)$ and L is less than ϵ.

In symbols:

$$\lim_{x \to a^+} f(x) = L \quad (2.4)$$

if and only if

$$\forall \epsilon > 0 \, \exists \delta > 0 \, (a < x < a + \delta \rightarrow |f(x) - L| < \epsilon) \quad (2.5)$$



These definitions state that no matter how small we choose ϵ to be, we can always find a small enough neighborhood of a such that the function $f(x)$ is always within ϵ units of the limit L, but only on one side of a.

Similarly, we say that the left-hand limit of $f(x)$ as x approaches a is L if for every positive number ϵ, there exists a corresponding positive number δ such that for all x satisfying $a - \delta < x < a$, the distance between $f(x)$ and L is less than ϵ.

In symbols:

$$\lim_{x \to a^-} f(x) = L \quad (2.6)$$

if and only if

$$\forall \epsilon > 0 \, \exists \delta > 0 \, (a - \delta < x < a \rightarrow \exists \delta > 0) \quad (2.7)$$

Example 2.3.3. Suppose we want to find the left-hand and right-hand limits of the function $f(x) = \frac{x^2-4}{x-2}$ as x approaches 2. We can consider the behavior of the function when x is slightly less than 2 and slightly greater than 2.

For the left-hand limit, we can substitute $x = 2-\delta$, where δ is a positive number approaching 0, into the function:

$$\lim_{x\to 2^-} f(x) = \lim_{\delta\to 0} \frac{(2-\delta)^2 - 4}{2-\delta-2} = -\infty$$

For the right-hand limit, we can substitute $x = 2 + \delta$, where δ is a positive number approaching 0, into the function:

$$\lim_{x\to 2^+} f(x) = \lim_{\delta\to 0} \frac{(2+\delta)^2 - 4}{2+\delta-2} = \infty$$

Therefore, the left-hand limit of $f(x)$ as x approaches 2 is $-\infty$, and the right-hand limit is ∞.

The left-hand, right-hand, and full (two-sided) limits of a function are related as follows: the left-hand and right-hand limits exist and are equal, if and only if the function has a limit at that point, which is equal to the common value of the left-hand and right-hand limits. This is the definition of a two-sided limit. That is:

$$\lim_{x\to a^-} f(x) = \lim_{x\to a^+} f(x) = L \quad \Leftrightarrow \quad \lim_{x\to a} f(x) = L \tag{2.8}$$

This means that if the function approaches the same value from both the left and right sides of the point, then it has a limit at that point, and the limit is equal to that common value.

However, if the left-hand and right-hand limits exist but are not equal, then the function does not have a limit at that point, and we say that the limit does not exist. This could happen if the function has a jump discontinuity, like a step function. That is:

$$\lim_{x\to a^-} f(x) \neq \lim_{x\to a^+} f(x) \quad \Rightarrow \lim_{x\to a} f(x) \text{ does not exist} \tag{2.9}$$

On the other hand, if either the left-hand or right-hand limit does not exist, then the full (two-sided) limit does not exist either. That is:

$$\lim_{x\to a^-} f(x) \text{ or } \lim_{x\to a^+} f(x) \text{ does not exist} \tag{2.10}$$

$$\Rightarrow \quad \lim_{x\to a} f(x) \text{ does not exist} \tag{2.11}$$

This means that if the function approaches different values from the left and right sides of the point, then it does not have a limit at that point.

For example, consider the function $f(x) = \frac{|x|}{x}$, which has a jump discontinuity at $x = 0$. The left-hand limit of $f(x)$ as x approaches 0 is -1, while the right-hand limit is 1. Therefore, the full (two-sided) limit of $f(x)$ as x approaches 0 does not exist.

2.3.1.2 Continuity

A function is said to be ***continuous*** at a point a if its limit exists at that point and is equal to the value of the function at that point. In other words, a function $f(x)$ is continuous at a if and only if

$$\lim_{x\to a} f(x) = f(a) \tag{2.12}$$

A function is said to be ***continuous on an interval*** if it is continuous at every point in that interval. Intuitively, a continuous function is one whose graph can be drawn without lifting the pen from the paper.

Example 2.3.4. Here's an example of a continuous function:

$f(x) = x^2$ is continuous on the interval $(-\infty, \infty)$, since the limit $\lim_{x\to a} x^2 = a^2$ exists for any real number a and is equal to $f(a)$.

Here's an example of a non-continuous function:

$$g(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}$$

$g(x)$ is not continuous at $x = 0$ because $\lim_{x\to 0^-} g(x) = 0$ and $\lim_{x\to 0^+} g(x) = 1$, which are not equal. This means that $g(x)$ has a "jump" or "discontinuity" at $x = 0$.

2.3.2 Limits to Infinity

A ***limit to infinity*** (also known as an *infinite limit* or a *limit at infinity*) is a special type of limit in calculus where we investigate the behavior of a function as the input approaches infinity. It tells us what happens to the function as the input gets larger and larger without bounds.

Limits to infinity are also an essential tool in calculus, especially when we want to investigate the long-term behavior of a function, such as its growth rate or whether it approaches a particular value as the input gets very large.

Formally, we say that the limit of a function $f(x)$ as x approaches infinity is L if we can make $f(x)$ arbitrarily close to L by making x sufficiently large. Formally, we write

$$\lim_{x\to\infty} f(x) = L \tag{2.13}$$

if and only if

$$\forall \epsilon > 0 \, \exists N > 0 \, ((x > N) \to (|f(x) - L| < \epsilon)) \tag{2.14}$$

Similarly, we can investigate the limit of a function as x approaches negative infinity if we can make $f(x)$ arbitrarily close to L by making x sufficiently negative. Formally, we write

$$\lim_{x\to-\infty} f(x) = L \tag{2.15}$$

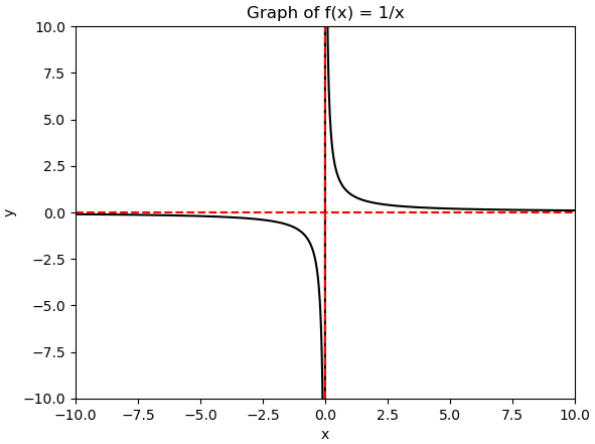
if and only if

$$\forall \epsilon > 0 \, \exists N > 0 \, ((x < N) \to (|f(x) - L| < \epsilon)) \tag{2.16}$$

Example 2.3.5. Consider the function $f(x) = \frac{1}{x}$. If we investigate the limit of $f(x)$ as x approaches infinity:

$$\lim_{x\to\infty} \frac{1}{x} = 0$$

This means that as x gets larger and larger, $\frac{1}{x}$ gets smaller and smaller, and approaches zero.



2.3.3 Limit Formulas

Here are the formulas for limits of basic arithmetic operations of functions:

$$\lim_{x\to a} c\cdot f(x) = c\cdot \lim_{x\to a} f(x) \tag{2.17}$$

$$\lim_{x\to a}(f(x)\pm g(x)) = \lim_{x\to a} f(x) \pm \lim_{x\to a} g(x) \tag{2.18}$$

$$\lim_{x\to a}(f(x)\cdot g(x)) = \lim_{x\to a} f(x)\cdot \lim_{x\to a} g(x) \tag{2.19}$$

$$\lim_{x\to a} \frac{f(x)}{g(x)} = \frac{\lim_{x\to a} f(x)}{\lim_{x\to a} g(x)} \tag{2.20}$$

$$\lim_{x\to a}(f(x))^r = [\lim_{x\to a} f(x)]^r \tag{2.21}$$

$$\lim_{x\to a} \sqrt[n]{f(x)} = \sqrt[n]{\lim_{x\to a} f(x)} \tag{2.22}$$

These formulas are very useful when computing limits of more complex functions that can be expressed in terms of basic arithmetic operations.

2.3.4 Evaluation Techniques

Here are some techniques that come in handy in calculating the limits of certain functions. (For L'Hospital's Rule, if needed, read the next section on derivatives first and then revisit the technique and its example.) They are all provable using the definitions of limits or derivatives; we omit proofs here.

2.3.4.1 Continuous Functions and Composition

If $f(x)$ and $g(x)$ are both continuous functions, then the limit of their composition as x approaches a value a is given by:

$$\lim_{x\to a} f(g(x)) = f\left(\lim_{x\to a} g(x)\right) \tag{2.23}$$

For example, suppose we want to find the limit of $\sin(x^2)$ as x approaches 0. We can use the fact that $\sin(x)$ is a continuous function and the above formula to find the limit:

$$\lim_{x\to 0} \sin(x^2) = \sin\left(\lim_{x\to 0} x^2\right) = \sin(0) = 0$$

2.3.4.2 Factoring and Cancelling

If we have a limit of the form $\frac{f(x)}{g(x)}$ where both $f(x)$ and $g(x)$ approach 0 or $\pm\infty$ as x approaches a value a, and $g(x)$ is not equal to 0 in a small neighborhood of a, then we can often simplify the expression by factoring and cancelling:

$$\lim_{x\to a} \frac{f(x)}{g(x)} = \lim_{x\to a} \frac{f(x)/h(x)}{g(x)/h(x)}$$

where $h(x)$ is a function that cancels out the common factors in the numerator and denominator. For example, suppose we want to find the limit of $\frac{x^2-4}{x-2}$ as x approaches 2. We can simplify the expression by factoring and cancelling:

$$\lim_{x\to 2} \frac{x^2-4}{x-2} = \lim_{x\to 2} \frac{(x-2)(x+2)}{x-2} = \lim_{x\to 2}(x+2) = 4$$

2.3.4.3 Rationalizing Numerator/Denominator

If we have a limit of the form $\frac{f(x)}{\sqrt[n]{g(x)}}$ where both $f(x)$ and $g(x)$ approach 0 or $\pm\infty$ as x approaches a value a, then we can often simplify the expression by rationalizing the numerator or denominator:

$$\lim_{x\to a} \frac{f(x)}{\sqrt[n]{g(x)}} = \lim_{x\to a} \frac{f(x)\cdot \sqrt[n]{g(x)}^{n-1}}{g(x)}$$

$$= \frac{\lim_{x\to a} f(x)\cdot \lim_{x\to a} \sqrt[n]{g(x)}^{n-1}}{\lim_{x\to a} g(x)}$$

For example, suppose we want to find the limit of $\frac{\sin(x)}{\sqrt{x}}$ as x approaches 0. We can simplify the expression by rationalizing the denominator:

$$\lim_{x\to 0}\frac{\sin(x)}{\sqrt{x}} = \lim_{x\to 0}\frac{\sin(x)\cdot\sqrt{x}}{x} = \lim_{x\to 0}\frac{\sin(x)}{x}\cdot\lim_{x\to 0}\sqrt{x}$$

$$= 1\cdot 0 = 0$$

2.3.4.4 Combining Rational Expressions

To find the following limit:

$$\lim_{x\to 1}\frac{x^2-1}{x-1}\cdot\frac{1}{x+1}$$

we can factor the numerator of the first expression as $(x+1)(x-1)$, which cancels with the denominator $(x-1)$ in the second expression. Thus, we have:

$$\lim_{x\to 1}\frac{x+1}{x+1} = 1$$

2.3.4.5 L'Hospital's/L'Hôpital's Rule

L'Hospital's rule is a mathematical technique used to evaluate limits that involve indeterminate forms such as $\frac{0}{0}$ or $\frac{\infty}{\infty}$. The general form of L'Hospital's rule is:

If $\lim_{x\to a} f(x) = 0$ and $\lim_{x\to a} g(x) = 0$ or $\lim_{x\to a} g(x) = \pm\infty$, then

$$\lim_{x\to a}\frac{f(x)}{g(x)} = \lim_{x\to a}\frac{f'(x)}{g'(x)}$$

where $f'(x)$ and $g'(x)$ are the derivatives of $f(x)$ and $g(x)$, respectively, evaluated at $x = a$.

If the limit on the right-hand side still involves an indeterminate form, we can apply L'Hospital's rule again until the limit can be evaluated. Note that L'Hospital's rule may not always be applicable or may require some algebraic manipulation of the original function before it can be used.

For example, suppose we want to calculate the following:

$$\lim_{x\to 0}\frac{e^x-1-x}{x^2}$$

We can apply L'Hopital's rule by taking the derivative of both the numerator and denominator with respect to x:

$$\lim_{x\to 0}\frac{e^x-1-x}{x^2} = \lim_{x\to 0}\frac{e^x-1}{2x} = \frac{1}{2}$$

2.4 Derivatives

The derivative of a function is a measure of how much the function changes at a particular point. It is defined as the instantaneous rate of change of the function at that point, or equivalently, as the slope of the tangent line to the function at that point. In other words, the derivative tells us how fast the function is changing and in what direction.

2.4.1 Basic Derivatives

The ***derivative*** of a function $f(x)$ at a point $x = a$ is denoted by $f'(a)$ or $\frac{df}{dx}(a)$. Geometrically, it is equal to the slope of the tangent line to the function at the point $(a, f(a))$.

Officially, it is defined as the limit of the difference quotient as the distance between x and a approaches zero:

$$f'(a) = \lim_{h\to 0}\frac{f(a+h)-f(a)}{h} \tag{2.24}$$

if this limit exists. Here, h is the change in x and $a+h$ is the point on the curve that is h units away from a.

Notations for the derivative of a function include:

$$f'(x) \equiv \frac{df}{dx}(x) \equiv \frac{dy}{dx} \equiv y' \equiv Df(x) \tag{2.25}$$

The notation used depends on the context and personal preference.

Example 2.4.1. We can use the definition of the derivative to find the derivative of the function $f(x) = 2x^2 + 3$:

$$\begin{aligned} f'(x) &= \lim_{h\to 0}\frac{f(x+h)-f(x)}{h} \\ &= \lim_{h\to 0}\frac{2(x+h)^2+3-2x^2-3}{h} \\ &= \lim_{h\to 0}\frac{2x^2+4xh+2h^2+3-2x^2-3}{h} \\ &= \lim_{h\to 0}\frac{4xh+2h^2}{h} = \lim_{h\to 0} 2(2x+h) = 4x \end{aligned}$$

Therefore, the derivative of $f(x) = 2x^2 + 3$ is $f'(x) = 4x$.

Here are some of the major properties of derivatives:[12]

1. *Derivative of a constant function.* The derivative of a constant function is zero:

$$\frac{d}{dx}(c) = 0 \tag{2.26}$$

where c is a constant.

2. *Derivative of a scalar multiple.* If a function $f(x)$ is multiplied by a constant c, then the derivative of the resulting function is equal to c times the derivative of the original function.

$$\frac{d}{dx}(cf(x)) = c\frac{d}{dx}(f(x)) \tag{2.27}$$

3. *Power rule.* If a function $f(x)$ is raised to a power n, then its derivative is given by n times the function raised to the power $n-1$, multiplied by the derivative of the function.

$$\frac{d}{dx}(f(x)^n) = n(f(x))^{n-1}\frac{d}{dx}(f(x)) \tag{2.28}$$

4. *Product rule.* If two functions $f(x)$ and $g(x)$ are multiplied together, then the derivative of the product is given by the first function times the derivative of the second function plus the second function times the derivative of the first function.

$$\frac{d}{dx}(f(x)g(x)) = f(x)\frac{d}{dx}(g(x)) + g(x)\frac{d}{dx}(f(x)) \tag{2.29}$$

5. *Quotient rule.* If two functions $f(x)$ and $g(x)$ are divided, then the derivative of the quotient is given by the numerator times the derivative of the denominator minus the denominator times the derivative of the numerator, divided by the denominator squared.

$$\frac{d}{dx}\left(\frac{f(x)}{g(x)}\right) = \frac{g(x)\frac{d}{dx}(f(x)) - f(x)\frac{d}{dx}(g(x))}{g(x)^2} \tag{2.30}$$

6. *Chain rule.* If a function $f(x)$ is composed with another function $g(x)$, then the derivative of the resulting function is given by the derivative of the outer function evaluated at the inner function, multiplied by the derivative of the inner function.

$$\frac{d}{dx}(f(g(x))) = \frac{df}{dg}(g(x)) \times \frac{dg}{dx} \tag{2.31}$$

These properties are used extensively in calculus to calculate the derivatives of functions.

Example 2.4.2. Suppose we have the function $f(x) = \sin(2x)$. We want to find the derivative $f'(x)$.

Using the chain rule, we can write:

$$f'(x) = \frac{d}{dx}\sin(2x) = \frac{d}{d(2x)}\sin(2x) \cdot \frac{d(2x)}{dx}$$

The derivative of $\sin(2x)$ with respect to $2x$ is $\cos(2x)$, so we can substitute that in:

$$f'(x) = \cos(2x) \cdot \frac{d(2x)}{dx}$$

The derivative of $2x$ with respect to x is simply 2, so we can substitute that in as well:

$$f'(x) = \cos(2x) \cdot 2$$

Simplifying, we get:

$$f'(x) = 2\cos(2x)$$

So the derivative of $f(x) = \sin(2x)$ is $f'(x) = 2\cos(2x)$.

2.4.2 Implicit Differentiation

Implicit differentiation is a technique used to find the derivative of an implicitly defined function, where the dependent variable is not explicitly written in terms of the independent variable. Instead, the equation defines a relationship between the variables that is not easily solvable for the dependent variable. Implicit equations often take the form of polynomial equations, transcendental equations, or systems of equations.

The idea of implicit differentiation is to use the chain rule to differentiate both sides of the equation with respect to the independent variable, treating the dependent variable as a function of the independent variable. This results in an equation that can be used to find the derivative of the implicitly defined function.

Here are some examples of implicit equations:

1. The equation of a circle, where x and y are the independent variables and r is a constant radius:
$$x^2 + y^2 = r^2$$
This equation defines a relationship between x and y that describes a circle of radius r centered at the origin.
2. The equation of an ellipse, where x and y are the independent variables and a and b are constants representing the semi-major and semi-minor axes, respectively:
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$
This equation defines a relationship between x and y that describes an ellipse centered at the origin.

3. The logistic function equation, used to model population growth, where t is the independent variable and N and r are constants:

$$N = \frac{K}{1 + e^{-rt}}$$

This equation defines a relationship between t and N that describes the growth of a population that approaches a limiting value K.

In each of these examples, the dependent variable is not explicitly defined in terms of the independent variable(s), but rather the equation defines a relationship between them.

Now let's do some implicit differentiation:

Example 2.4.3. Let's find the derivative of the circle equation $x^2 + y^2 = 25$ with respect to x. We start by differentiating both sides of the equation with respect to x:

$$\begin{aligned} \frac{d}{dx}(x^2 + y^2) &= \frac{d}{dx}(25) \\ \frac{d}{dx}(x^2) + \frac{d}{dx}(y^2) &= 0 \\ 2x + 2y\frac{dy}{dx} &= 0 \end{aligned}$$

We can then solve for $\frac{dy}{dx}$:

$$\begin{aligned} 2x + 2y\frac{dy}{dx} &= 0 \\ \frac{dy}{dx} &= -\frac{x}{y} \end{aligned}$$

Example 2.4.4. Find the derivative of the equation $x^3 + y^3 = 9xy$ with respect to x. Again, we differentiate both sides of the equation with respect to x:

$$\begin{aligned} \frac{d}{dx}(x^3 + y^3) &= \frac{d}{dx}(9xy) \\ 3x^2 + 3y^2\frac{dy}{dx} &= 9y + 9x\frac{dy}{dx} \end{aligned}$$

We can then solve for $\frac{dy}{dx}$:

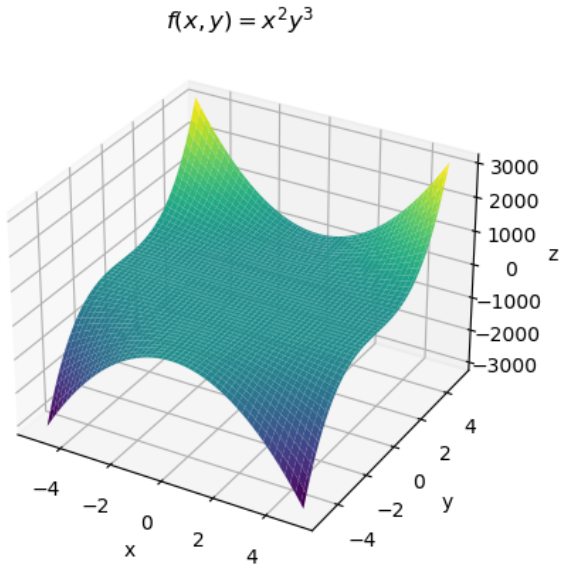
$$\begin{aligned} 3x^2 + 3y^2\frac{dy}{dx} &= 9y + 9x\frac{dy}{dx} \\ 3y^2\frac{dy}{dx} - 9x\frac{dy}{dx} &= 9y - 3x^2 \\ \frac{dy}{dx} &= \frac{9y - 3x^2}{3y^2 - 9x} \end{aligned}$$

So the derivative of the equation $x^3 + y^3 = 9xy$ with respect to x is $\frac{dy}{dx} = \frac{9y-3x^2}{3y^2-9x}$.

2.4.3 Partial and Total Derivatives

Partial derivative is a measure of how a function changes when one of its input variables changes, while holding the other variables constant. Partial derivatives are used to analyze the behavior of functions of two or more variables, and are an important tool in many areas of mathematics and science.

Example 2.4.5. Find the partial derivatives of the function $f(x, y) = x^2y^3$.



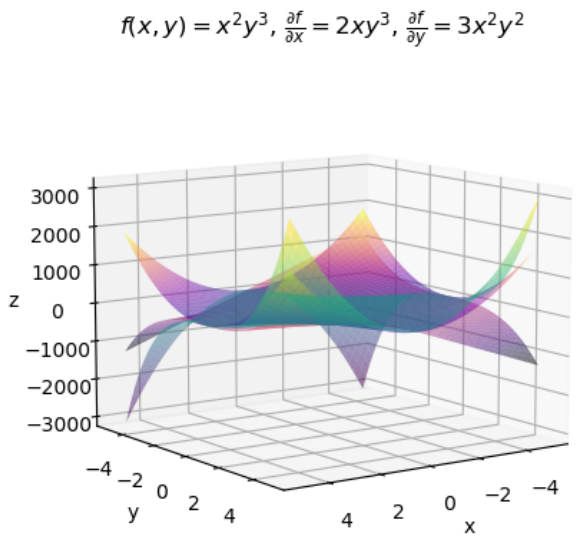
To find the partial derivative of f with respect to x, we treat y as a constant and differentiate f with respect to x using the power rule:

$$\frac{\partial f}{\partial x} = 2xy^3.$$

To find the partial derivative of f with respect to y, we treat x as a constant and differentiate f with respect to y using the power rule:

$$\frac{\partial f}{\partial y} = 3x^2y^2.$$

These partial derivatives describe how the function f changes when x or y changes, while holding the other variable constant.

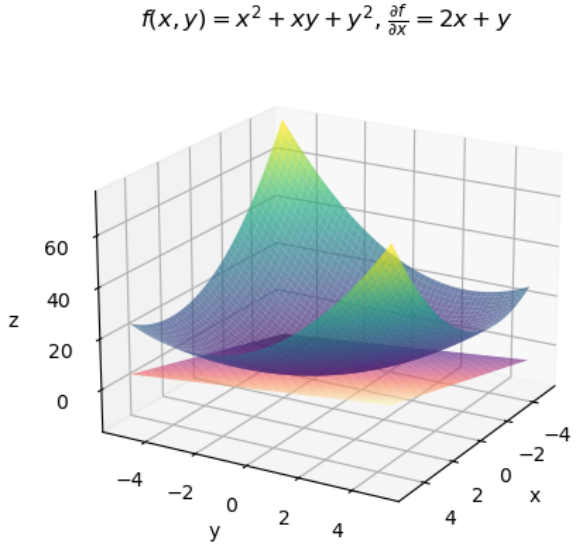


Example 2.4.6. Find the partial derivative of the function $f(x,y) = x^2 + xy + y^2$ with respect to x.

To find the partial derivative of f with respect to x, we treat y as a constant and differentiate f with respect to x using the power rule:

$$\frac{\partial f}{\partial x} = 2x + y.$$

This represents the infinitesimal change in the variable y when we take an infinitesimal change in x.[13] This tells us that when we change x by a very small amount, f changes by approximately $2x + y$ times that amount, while y stays fixed.



Now, ***total derivatives***, also known as *full derivatives*, are a type of derivative that take into account changes in *all* independent variables of a function. They are used in multivariable calculus to describe how a function changes as all of its input variables change simultaneously.

The total derivative of a function $f(x_1, x_2, \ldots, x_n)$ with respect to x_i is denoted by $\frac{df}{dx_i}$ or $D_i f$, and is defined as:

$$D_i f = \frac{\partial f}{\partial x_1}dx_1 + \frac{\partial f}{\partial x_2}dx_2 + \cdots + \frac{\partial f}{\partial x_n}dx_n \tag{2.32}$$

where dx_i represents an infinitesimal change in the variable x_i. The total derivative describes the instantaneous rate of change of a function with respect to each of its input variables.

Here are two examples of total derivatives:

Example 2.4.7. Find the total derivative of the function $f(x,y) = x^2y^3$ with respect to x.

Using the total derivative formula, we have:

$$D_1 f = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy$$

where dx and dy are the infinitesimal changes in x and y, respectively. Taking the partial derivatives, we have:

$$D_1 f = 2xy^3dx + 3x^2y^2dy$$

So the total derivative of f with respect to x is $D_1 f = 2xy^3dx + 3x^2y^2dy$.

Example 2.4.8. Find the total derivative of the function $f(x,y,z) = x^2 + y^2 + z^2$ with respect to z.

Using the total derivative formula, we have:

$$D_3 f = \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial z}dz$$

Taking the partial derivatives, we have:

$$D_3 f = 0 + 0 + 2zdz$$

So the total derivative of f with respect to z is $D_3 f = 2zdz$. This tells us that if we increase z by a small amount dz, the value of f will increase by $2zdz$.

2.4.4 Higher Derivatives

Higher derivatives refer to the derivatives of a function with respect to its independent variable, taken more than once. For example, the first derivative of a function is the rate of change of the function with respect to its independent variable, while the second derivative is the rate of change of the first derivative with respect to the independent variable.

Higher derivatives can be obtained from first-order derivatives by taking successive derivatives.

For example, the second derivative can be obtained by taking the derivative of the first derivative, and the third derivative can be obtained by taking the derivative of the second derivative, and so on.

The typical notation for higher derivatives involves the use of prime symbols. For example, the first derivative of a function $f(x)$ can be denoted as $f'(x)$, the second derivative can be denoted as $f''(x)$, and so on.

The ***gradient*** of a function is the vector of its partial derivatives and is typically denoted using the symbol ∇. The first derivative of a function $f(x, y)$ with respect to x can be denoted as $\frac{\partial f}{\partial x}$, and the gradient of $f(x, y)$ can be denoted as:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) \tag{2.33}$$

If you have a function with an n-dimensional input $\vec{x}$ and m-dimensional output $\vec{y} = f(\vec{x})$, the complete gradient is a matrix of the derivative of every output with respect to every input, called the Jacobian.

Formally, the ***Jacobian*** matrix is defined as:

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix} \tag{2.34}$$

Each element in the Jacobian matrix is the partial derivative of the corresponding output with respect to the corresponding input. For example, the element in the ith row and jth column of the Jacobian matrix is $\frac{\partial y_i}{\partial x_j}$, which is the partial derivative of the ith output with respect to the jth input.

The Jacobian matrix is useful in many areas of mathematics and science, including optimization, control theory, and machine learning. It provides information about the rate of change of a function with respect to its inputs, and can be used to solve optimization problems and analyze the stability of systems

Now, the ***Hessian matrix*** is a square matrix of *second-order* partial derivatives of a function with respect to its variables. If the function f has n variables, then the Hessian matrix is an $n \times n$ matrix whose (i, j) entry is the second partial derivative of f with respect to the i-th and j-th variables, written as $\frac{\partial^2 f}{\partial x_i \partial x_j}$.

The Hessian matrix can be written as:

$$\mathrm{H}(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.35}$$

The Hessian matrix provides information about the curvature of a function. In particular, the sign of the eigenvalues of the Hessian matrix can be used to determine whether a critical point of the function is a maximum, a minimum, or a saddle point.[14] The Hessian matrix is also used in optimization algorithms to find local extrema of a function.

Example 2.4.9. The second derivative of the function $f(x) = x^3$:

$$\begin{aligned} f(x) &= x^3 \\ f'(x) &= 3x^2 \\ f''(x) &= \frac{d}{dx}(3x^2) = 6x \end{aligned}$$

Example 2.4.10. The gradient and Hessian of the function $f(x, y) = x^2 + 2xy + y^2$ is obtained as follows:

$$\nabla f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right) = (2x + 2y, 2x + 2y)$$

$$\text{Hessian}(f) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 2 & 2 & 2 & 2 \end{bmatrix}$$

2.4.5 The Shape of a Function

In this section, we discuss how to determine certain behaviours of a function based on its derivatives.

2.4.5.1 Increasing and Decreasing

To determine whether a function is increasing or decreasing on a certain interval, we can use the first derivative test. Here are the steps:

1. Take the derivative of the function. Find the critical points by solving $f'(x) = 0$ or $f'(x)$ does not exist.
2. Create a sign chart for $f'(x)$ on the interval of interest.
3. Determine the sign of $f'(x)$ in each interval between the critical points.
4. If $f'(x)$ is positive in an interval, the function is increasing on that interval. If $f'(x)$ is negative in an interval, the function is decreasing on that interval.

Back to our last example: to determine where the function $f(x) = x^3 - 3x^2 + 2$ is increasing or decreasing, we first find its derivative:

$$f'(x) = 3x^2 - 6x$$

Next, we find the critical points by solving $f'(x) = 0$ or $f'(x)$ does not exist:

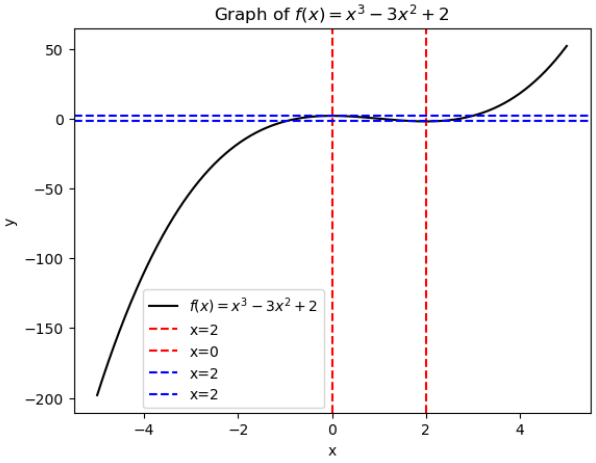
$$f'(x) = 0 \quad \Rightarrow \quad x = 0, 2$$

Note that $f'(x)$ exists for all values of x, so we do not have to worry about cases where $f'(x)$ does not exist.

Now we can create a sign chart for $f'(x)$:

Interval	$(-\infty, 0)$	$(0, 2)$	$(2, \infty)$
$f'(x)$	+	−	+
$f(x)$	increasing	decreasing	increasing

Therefore, we can conclude that $f(x)$ has a relative minimum at $x = 0$ and a relative maximum at $x = 2$. Additionally, $f(x)$ is increasing on the intervals $(-\infty, 0)$ and $(2, \infty)$, and decreasing on the interval $(0, 2)$.



2.4.5.2 Convex and Concave

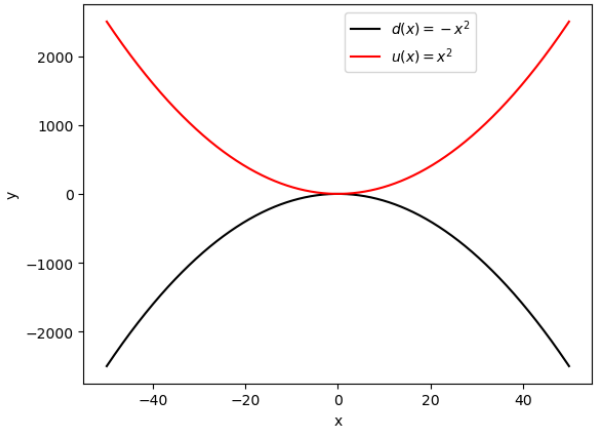
A function is said to be ***convex*** if its graph curves upward, i.e., if the line segment connecting any two points on the graph lies above or on the graph itself. A function is said to be ***concave*** if its graph curves downward, i.e., if the line segment connecting any two points on the graph lies below or on the graph itself.

For the function $u(x) = x^2$, its graph is a parabola that curves upward, which means it is a convex function. To see this, we can look at the second derivative:

$$u''(x) = 2$$

Since $u''(x) > 0$ for all x, we know that $u(x)$ is a convex function. Another way to see this is to note that the graph of $u(x)$ lies entirely above its tangent lines.

Similarly, it can be shown that $d(x) = -x^2$ is a concave function.

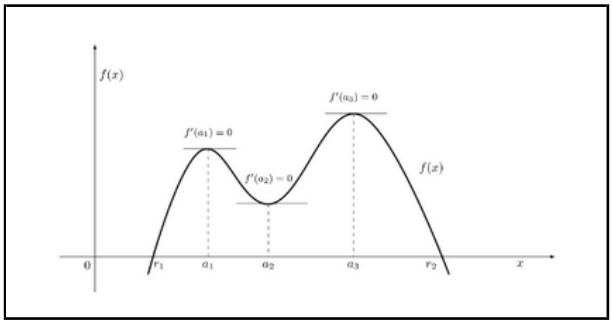


2.4.6 Finding the Extrema

In this final section, we discuss several systematic ways to find extreme values of functions.

2.4.6.1 Critical Points and Extrema

In calculus, an ***absolute maximum or minimum*** is the largest or smallest value that a function attains over its entire domain. A ***local maximum or minimum*** is the largest or smallest value that a function attains in a small neighborhood around a specific point in its domain. ***Critical*** points are points where the derivative of a function is zero or undefined.



For example, critical points of the function $f(x)$ illustrated above are $x = a_1, x = a_2$ and $x = a_3$. You can use the critical points to find the location of a function's maxima and minima.

To find the critical points of f, we first find its derivative $f'(x)$ and then solve the equation $f'(x) = 0$ or find the values of x where $f'(x)$ is undefined.

Once we have found the critical points of f, we can use the first or second derivative test to determine whether each critical point corresponds to a local maximum, local minimum, or neither. The first derivative test states that if $f'(x_0) = 0$ and $f''(x_0) < 0$, then $f(x_0)$ is a local maximum; if $f'(x_0) = 0$ and $f''(x_0) > 0$, then $f(x_0)$ is a local

minimum; and if $f'(x_0) = 0$ and $f''(x_0) = 0$, then the test is inconclusive and we must use other methods to determine whether $f(x_0)$ is a local maximum, local minimum, or neither.

To find the absolute maximum and minimum of f over its entire domain, we need to consider not only its critical points but also its values at the endpoints of its domain. Specifically, we compute $f(x)$ for all critical points and endpoints, and then compare those values to find the absolute maximum and minimum.

Example 2.4.11. Consider, again, the function $f(x) = x^3 - 3x^2 + 2$. Its derivative is $f'(x) = 3x^2 - 6x$, and its second derivative is $f''(x) = 6x - 6$. To find the critical points, we solve the equation $f'(x) = 0$:

$$f'(x) = 3x^2 - 6x = 0 \implies x(x-2) = 0 \implies x = 0, 2$$

Therefore, the critical points of f are $x = 0$ and $x = 2$. To determine whether these points correspond to local maximum or minimum, we use the first derivative test. At $x = 0$, we have $f'(0) = 0$

$$f'(0) = 0 \quad f''(0) = 6(0) - 6 = -6$$

Since $f'(0) = 0$ and $f''(0) < 0$, the first derivative test tells us that $f(0)$ is a local maximum. Similarly, at $x = 2$ we have:

$$f'(2) = 0 \quad f''(2) = 6(2) - 6 = 6$$

Since $f'(2) = 0$ and $f''(2) > 0$, the first derivative test tells us that $f(2)$ is a local minimum. You can also visually confirm these using the graph provided above.

2.4.6.2 Fermat's Theorem

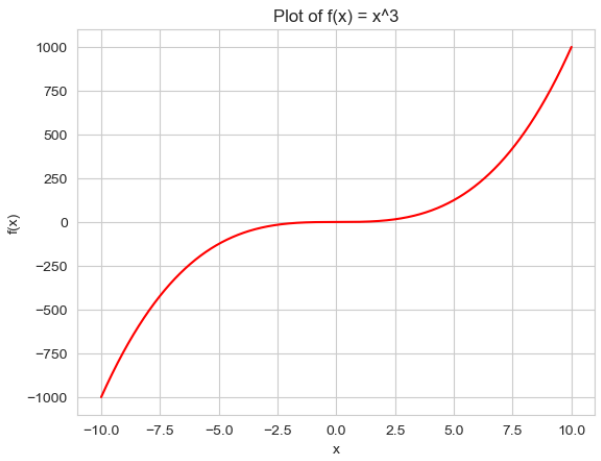
Fermat's Theorem, also known as *First Derivative Test*, is a fundamental theorem in calculus that describes the behavior of continuous functions on closed intervals.

Theorem 2.4.1 (Fermat's Theorem)**.** If a function $f(x)$ has a local maximum or minimum at $x = c$, and $f(x)$ is differentiable at $x = c$, then $f'(c) = 0$.

The converse of Fermat's Theorem is not true, which means that $f'(c) = 0$ at a point c does not necessarily imply that $f(x)$ has a local extremum at $x = c$.

Example 2.4.12. One classic example where the converse of Fermat's Theorem is not true is the function $f(x) = x^3$. This function has a critical point at $x = 0$, since $f'(x) = 3x^2$ and $f'(0) = 0$. However, $f(x)$ does not have a local maximum or minimum at $x = 0$.

To see why, we can examine the behavior of $f(x)$ near $x = 0$. When $x < 0$, $f(x) = x^3$ is negative and decreasing. When $x > 0$, $f(x) = x^3$ is positive and increasing. Therefore, $f(x)$ has no local maximum or minimum at $x = 0$, despite having a critical point there. This example illustrates that the converse of Fermat's Theorem is not true in general, and that additional analysis is often necessary to determine whether a critical point corresponds to a local extremum of a function.



2.4.6.3 Extreme Value Theorem

The Extreme Value Theorem is a powerful result that is used frequently in calculus to analyze the behavior of functions on closed and bounded intervals. It states that a continuous function defined on a closed and bounded interval must attain both its *maximum and minimum* (i.e., absolute extrema) values at some point within that interval.

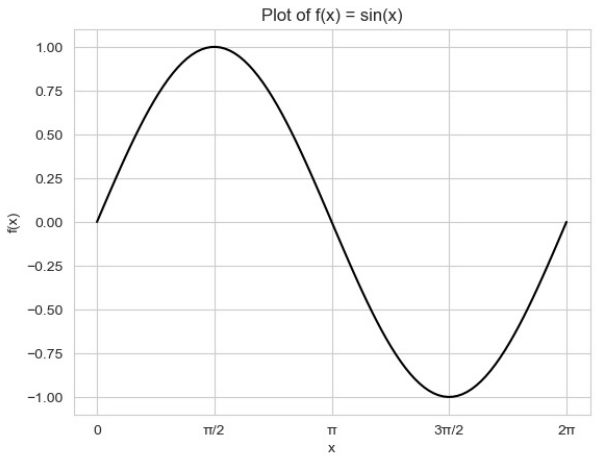
Theorem 2.4.2 (Extreme Value Theorem)**.** Let $f(x)$ be a continuous function on the closed and bounded interval $[a, b]$. Then there exist values c_1 and c_2 in $[a, b]$ such that:

$$f(c_1) \geq f(x) \geq f(c_2) \quad \text{for all } x \in [a, b] \tag{2.36}$$

where $f(c_1)$ is the maximum value of f on $[a, b]$ and $f(c_2)$ is the minimum value of f on $[a, b]$.

Example 2.4.13. Suppose we have a continuous function $f(x) = \sin(x)$ defined on the closed and bounded interval $[0, \pi]$. To apply the Extreme Value Theorem, we note that f is continuous on $[0, \pi]$, which is a closed and bounded interval. Therefore, by the Extreme Value Theorem, $f(x)$ must attain both its maximum and minimum values on $[0, \pi]$.

To find these values, we can use the fact that $\sin(x)$ attains its maximum value of 1 when $x = \frac{\pi}{2}$, and its minimum value of -1 when $x = \frac{3\pi}{2}$. Therefore, we can conclude that the maximum value of f on $[0, \pi]$ is $f(\frac{\pi}{2}) = 1$, and the minimum value of f on $[0, \pi]$ is $f(\frac{3\pi}{2}) = -1$.



2.4.6.4 Mean Value Theorem

The ***Mean Value Theorem*** relates the values of a function to its derivative. Informally, it states that if a function is continuous on a closed interval and differentiable on the open interval, then there must be a point within the interval where the slope of the tangent line is equal to the slope of the secant line connecting the endpoints of the interval. More formally:

Theorem 2.4.3 (Mean Value Theorem)**.** Let $f(x)$ be a function that is continuous on the closed interval $[a, b]$ and differentiable on the open interval (a, b). Then there exists a point c in the open interval such that:

$$f'(c) = \frac{f(b) - f(a)}{b - a} \tag{2.37}$$

In other words, there exists a point c such that the slope of the tangent line to $f(x)$ at c is equal to the slope of the secant line connecting the points $(a, f(a))$ and $(b, f(b))$.

Example 2.4.14. Suppose we want to find a point on the interval $[1, 3]$ where the slope of the tangent line to $f(x) = x^2$ is equal to the slope of the secant line connecting $(1, 1)$ and $(3, 9)$. First, we compute the slope of the secant line:

$$\frac{f(3) - f(1)}{3 - 1} = \frac{9 - 1}{2} = 4$$

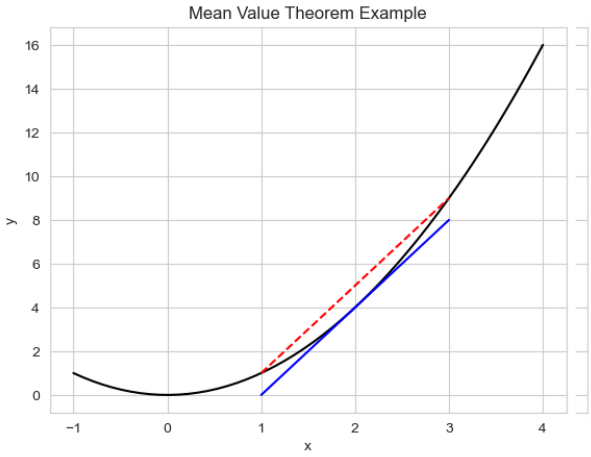
Now, we need to find a point c on the interval $(1, 3)$ where the slope of the tangent line to $f(x)$ is also equal to 4. We take the derivative of $f(x)$ to find:

$$f'(x) = 2x$$

Setting $f'(x) = 4$, we solve for x to get:

$$2x = 4 \Rightarrow x = 2$$

Therefore, by the Mean Value Theorem, there exists a point c in the interval $(1, 3)$ where $f'(c) = 4$, and we have found that point to be $c = 2$.



2.4.6.5 Newton's Method

Newton's method, also known as the *Newton-Raphson method*, is an iterative numerical method for finding the roots of a differentiable function.

The method starts with an initial guess x_0 for the root of the function $f(x)$ and then refines the guess using the formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.38}$$

where $f'(x_n)$ is the first derivative of $f(x)$ evaluated at x_n. This formula gives an improved estimate x_{n+1} of the root, which is then used as the new starting point for the next iteration. The process is repeated until the desired level of accuracy is achieved.

The method can be written in the following general form:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2.39}$$

where x_n is the nth approximation of the root, $f(x_n)$ is the function evaluated at x_n, and $f'(x_n)$ is the first derivative of the function evaluated at x_n.

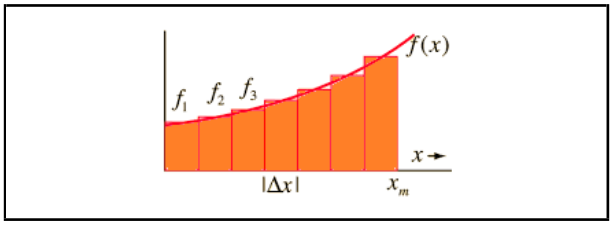
2.5 Integrals

Now that we have learned about limits and derivatives, we have the background to understand integrals. ***Integrals*** are mathematical tools that allow us to find the area under a curve or the total accumulation of a quantity over an interval. They are closely related to derivatives, as the derivative of a function represents its instantaneous rate of change, while the integral represents its cumulative change over a given interval.

The Fundamental Theorem of Calculus establishes the relationship between integrals and derivatives, showing that the integral of a function can be computed by finding its 'antiderivative' and evaluating it at the endpoints of the interval (see below for a rigorous representation). By learning about integrals, we gain a deeper understanding of the behavior of functions and their applications in a variety of fields, including physics, engineering, and economics.

To understand the connection between integrals and areas, let's consider a simple example. Suppose we have a function $y = f(x)$ that represents the height of a curve above the x-axis. We want to find the area under the curve between two values of x, say a and b.

One way to do this is to divide the area into small rectangles, as shown in the figure below:



The width of each rectangle is Δx, and the height is given by the function $f(x)$ at some point within the rectangle. As we make the width of the rectangles smaller and smaller, the approximation becomes better and better, and in the limit as the width goes to zero, the approximation becomes exact.

The area under the curve between a and b is then given by the sum of the areas of all these rectangles:

$$\text{Area} = \sum_{i=1}^{n} f(x_i)\Delta x \tag{2.40}$$

As we make the width of the rectangles smaller, we take the limit as n goes to infinity and Δx goes to zero. This leads us to the rigorous definition of integrals.

The integral of a function $f(x)$ over an interval $[a, b]$ is defined as the limit of a sum of rectangles, as the width of the rectangles goes to zero. This is expressed mathematically as:

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i)\Delta x \tag{2.41}$$

Here, dx represents an infinitesimal width, and the limit of the sum represents the exact area under the curve.

The relationship between integrals and derivatives is given by the Fundamental Theorem of Calculus, as follows:

Theorem 2.5.1 (Fundamental Theorem of Calculus)**.** If $F(x)$ is an antiderivative of $f(x)$, meaning that $F'(x) = f(x)$ then the definite integral of $f(x)$ over $[a, b]$ is given by

$$\int_a^b f(x)\,dx = F(b) - F(a) \tag{2.42}$$

This means that we can use derivatives to compute integrals, and vice versa. For example, if we want to find the area under a curve, we can find an antiderivative of the function and evaluate it at the endpoints of the interval to get the exact area.

Note that sometimes it's common to use the notation $\int_a^b f(x)\,dx = F(x)\Big|_a^b$ in calculating definite integrals.

Example 2.5.1. Suppose we want to compute the following integral:

$$\int_a^b x^2\,dx$$

Using the Fundamental Theorem of Calculus, we can find an antiderivative of the integrand x^2, which is $\frac{x^3}{3}$. Then, we can evaluate the definite integral by computing the difference between the antiderivative evaluated at the endpoints of the interval:

$$\int_0^2 x^2 dx = \frac{d}{dx}\left(\frac{x^3}{3}\right)\Big|_0^2 = \left(\frac{2^3}{3}\right) - \left(\frac{0^3}{3}\right) = \frac{8}{3}$$

So the value of the definite integral from 0 to 2 of the function x^2 is $\frac{8}{3}$.

We suffice this much for integrals as they're not used much later and this basic understanding and intuition will help with the material in the rest of the book.

Notes

[11]The following two images are adopted from https://blogs.scientificamerican.com/roots-of-unity/the-subterfuge-of-epsilon-and-delta/

[12]Strictly speaking, we could introduce the chain rule and then show how the *product* and *quotient* are special cases. Also, the quotient rule is a special case of the product rule when we note that $\frac{1}{x} = x^{-1}$.

[13]'Infinitesimal' refers to a quantity that is extremely small and can be thought of as approaching zero. In calculus, infinitesimals are used to represent changes in variables that are so small that they can be approximated as being instantaneous. For example, if we are taking the total derivative of a function $f(x, y)$ with respect to x, then dx represents an infinitesimal change in the variable x, and dy represents an infinitesimal change in the variable y. The total derivative tells us how much the function changes when both x and y change by infinitesimally small amounts dx and dy, respectively. The use of infinitesimals is a key concept in calculus, and allows us to describe how functions change in response to small variations in their inputs.

[14]If all eigenvalues are positive, then the critical point is a local minimum. If all eigenvalues are negative, then the critical point is a local maximum. If the eigenvalues have both positive and negative values, then the critical point is a saddle point.

2.6 Further Reading

Here are a few useful resources on Support Vector Machines:

1. *Calculus: Early Transcendentals*, by James Stewart [38]. This textbook provides a comprehensive introduction to calculus, covering functions, limits, derivatives, integrals, sequences, and series. It also includes discussions of multivariable calculus, including partial derivatives, directional derivatives, and gradients.
2. *Calculus: Early Transcendentals*, by William Briggs and Lyle Cochran [8]. This textbook covers the same topics as Stewart's *Calculus*, but is written in a slightly different style. It emphasizes the use of technology (such as graphing calculators) to aid in understanding and problem-solving.
3. *Introduction to Real Analysis*, by William Trench [43]. This is a more advanced textbook that covers calculus and real analysis, including topics such as sequences and series, continuity, differentiability, and integration. It is more theoretical than the other textbooks on this list, and is intended for advanced undergraduate or graduate students.
4. *Multivariable Calculus*, by James Stewart [39]. This textbook covers topics in multivariable calculus, including partial derivatives, gradients, directional derivatives, and optimization techniques. It is a good reference for anyone interested in these topics specifically.

Chapter 3

Probability Theory I: The Basics

Probability theory is a branch of mathematics that deals with the study of random events and their likelihood of occurrence. It provides a framework for reasoning about uncertainty and making predictions based on incomplete or noisy data.

In data science, probability theory is essential because it allows us to model and analyze complex systems that involve randomness or uncertainty. For example, we can use probability theory to estimate the likelihood of a particular outcome or event based on observed data, model the behavior of complex systems, such as financial markets or natural phenomena, using probabilistic models, and design and evaluate statistical algorithms and machine learning models that make use of probability distributions to make predictions.

Probability theory also provides the foundation for many statistical techniques used in data science and data analysis, such as hypothesis testing, confidence intervals, and Bayesian inference. By understanding the principles of probability theory, data scientists can make more accurate predictions, improve their models, and make better decisions based on data.

3.1 Unconditional Probability

Unconditional probability or ***Basic probability*** refers to the probability of an event without any prior knowledge or conditioning on other events. It is also known as *prior probability*.

The formula for basic probability is:

$$P(A) = \frac{\text{Number of favorable outcomes}}{\text{Total number of possible outcomes}} \tag{3.1}$$

where $P(A)$ is the probability of event A.

Example 3.1.1. Suppose we have a bag with 5 red balls and 3 blue balls. What is the probability of picking a red ball?

$$P(\text{Red}) = \frac{5}{8} = 0.625$$

In this example, the number of favorable outcomes is 5 (the number of red balls), and the total number of possible outcomes is 8 (the total number of balls in the bag). Therefore, the probability of picking a red ball is $\frac{5}{8}$ or 0.625.

3.2 Conditional Probability

Conditional probability refers to the probability of an event occurring given that another event has already occurred. It is a way of measuring the probability of an event in light of some additional information or context.

Intuitively, conditional probability can be thought of as answering the question: "What is the probability of event A happening, given that we know event B has already occurred?"

The formula for conditional probability is:

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)} \tag{3.2}$$

where $P(A|B)$ is the probability of event A happening given that event B has already occurred,

$P(A \text{ and } B)$ is the probability of both events A and B happening, and $P(B)$ is the probability of event B happening.

The ***Law of Total Probability*** states that the probability of an event A occurring can be calculated by considering all possible ways that A can occur, given a set of mutually exclusive and exhaustive events $B_1, B_2, B_3, ..., B_n$.

Mathematically, the law of total probability can be expressed as follows:

$$P(A) = \sum_{i=1}^{n} P(A|B_i)P(B_i) \tag{3.3}$$

Example 3.2.1. Suppose, again, that we have a bag with 5 red balls and 3 blue balls. We draw one ball at random and record its color. What is the probability of drawing a red ball on the second draw, given that the first ball was red? Suppose we don't return the first ball back to the pool.

Let A be the event of drawing a red ball on the second draw, and let B be the event of drawing a red ball on the first draw. We know from the previous example that the probability of drawing a red ball on the first draw is $\frac{5}{8}$. To calculate the probability of drawing a red ball on the second draw given that the first ball was red, we use the conditional probability formula:

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)}$$

$P(A \text{ and } B)$ is the probability of drawing a red ball on both the first and second draw, which is the same as the probability of drawing a red ball on the first draw multiplied by the probability of drawing a red ball on the second draw given that the first draw was red and the ball wasn't replaced:

$$P(A \text{ and } B) = \frac{5}{8} \times \frac{4}{7} = 0.35$$

$P(B)$ is the probability of drawing a red ball on the first draw, which we know is $\frac{5}{8}$. Therefore, we have:

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)} = \frac{0.35}{\frac{5}{8}} = 0.56$$

So the desired probability is 56%.

Example 3.2.2. Suppose that we have a group of 100 people, 60 of whom are men and 40 of whom are women. We also know that 30 of the men are over 50 years old, and 10 of the women are over 50 years old. What is the probability that a person chosen at random is over 50 years old, given that the person is a man?

Let A be the event of the person being over 50 years old, and let B be the event of the person being a man. We want to calculate $P(A|B)$, the probability of the person being over 50 years old given that the person is a man. We know that $P(B)$, the probability of the person being a man, is 0.6. To calculate $P(A \text{ and } B)$, the probability of the person being both over 50 years old and a man, we can use the multiplication rule:

$$P(A \text{ and } B) = P(B) \times P(A|B)$$

We know that $P(A|B)$, the probability of the person being over 50 years old given that the person is a man, is $\frac{30}{60} = 0.5$. Therefore:

$$P(A \text{ and } B) = 0.6 \times 0.5 = 0.3$$

To calculate $P(A)$, the probability of the person being over 50 years old, we can use the law of total probability:

$$P(A) = P(A|B) \times P(B) + P(A|\neg B) \times P(\neg B)$$

where $\neg B$ represents the event of the person not being a man. We know that $P(A|\neg B)$, the probability of the person being over 50 years old given that the person is not a man, is $\frac{10}{40} = 0.25$. Therefore:

$$P(A) = 0.5 \times 0.6 + 0.25 \times 0.4 = 0.35$$

Finally, we can use the formula for conditional probability to calculate $P(A|B)$:

$$P(A|B) = \frac{P(A \text{ and } B)}{P(B)} = \frac{0.3}{0.6} = 0.5$$

So the probability that a person chosen at random is over 50 years old, given that the person is a man, is 0.5 or 50%.

3.2.1 Bayes' Theorem

Bayes' Theorem is a fundamental concept in probability theory that relates conditional probabilities. It provides a way to update our beliefs or knowledge about an event based on new evidence or information.

Bayes' theorem is widely used in statistics, machine learning, and other fields where we need to update our beliefs about a hypothesis or model based on new evidence. It allows us to calculate the probability of a hypothesis given some observed data, by inverting the conditional probabilities that relate the hypothesis to the data.

Here's the formulation of the theorem:

Theorem 3.2.1 (Bayes' Theorem)**.** The probability of an event A given an event B is equal to the probability of event B given event A, multiplied by the prior probability of event A, divided by the prior probability of event B:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \tag{3.4}$$

By rearranging (3.4) we get the equality of conditional probabilities:

$$P(A|B) \times P(B) = P(B|A) \times P(A) \tag{3.5}$$

This equation shows that the probability of A occurring given that B has occurred is proportional to the probability of B occurring given that A has occurred, multiplied by the prior probability of A, and divided by the marginal probability of B.

To better understand Bayes' Theorem, let's look at an example.

Example 3.2.3. Suppose there is a rare disease that affects 1% of the population. A medical test has been developed that is 95% accurate, meaning that if a person has the disease, the test will correctly detect it 95% of the time, and if a person does not have the disease, the test will correctly identify them as not having the disease 95% of the time.

Now suppose you take the test and it comes back positive. What is the probability that you actually have the disease?

Using Bayes' Theorem, we can calculate the posterior probability of having the disease given a positive test result:

$$P(\text{disease}|\text{positive}) = \frac{P(\text{positive}|\text{disease}) \times P(\text{disease})}{P(\text{positive})}$$

We know that $P(\text{disease}) = 0.01$, the prior probability of having the disease, and $P(\text{positive}|\text{disease}) = 0.95$, the likelihood of testing positive given that the person has the disease. To calculate $P(\text{positive})$, the total probability of testing positive, we need to consider two cases:

1. The person has the disease and tests positive: probability = $P(\text{positive}|\text{disease}) \times P(\text{disease}) = 0.95 \times 0.01 = 0.0095$
2. The person does not have the disease but tests positive: probability = $P(\text{positive}|\text{no disease}) \times P(\text{no disease}) = 0.05 \times 0.99 = 0.0495$

Therefore, $P(\text{positive}) = 0.0095 + 0.0495 = 0.059$. Now we can substitute these values into the Bayes' theorem formula:

$$P(\text{disease}|\text{positive}) = \frac{0.95 \times 0.01}{0.059} = 0.161$$

So the probability of actually having the disease given a positive test result is 16.1%, which is much lower than the initial probability of 1%. This example shows how Bayes' Theorem can be used to update our beliefs about the probability of an event based on new evidence.

3.3 Independent and Mutually Exclusive Events

Independent events are events where the occurrence of one event does not affect the probability of the occurrence of the other event. In other words, the probability of both events occurring is simply the product of the probabilities of each event occurring separately.

Mathematically, two events A and B are independent if:

$$P(A \cap B) = P(A) \cdot P(B) \tag{3.6}$$

where $P(A)$ and $P(B)$ are the probabilities of events A and B occurring, respectively, and $P(A \cap B)$ is the probability of both A and B occurring.

For example, flipping a coin twice constitutes an independent event. The probability of getting heads on the first flip is 0.5, and the probability of getting heads on the second flip is also 0.5. The probability of getting heads on both flips is:

$$P(\text{heads on both flips}) = P(\text{heads on first flip}) \times P(\text{heads on second flip}) = 0.5 \times 0.5 = 0.25$$

Mutually exclusive events are events that cannot occur at the same time. Mathematically, two events A and B are mutually exclusive if:

$$P(A \cap B) = 0 \tag{3.7}$$

In other words, if A occurs, then B cannot occur, and vice versa. For example, rolling a die and getting a 3 and rolling a die and getting a 4 are mutually exclusive events because it is impossible for both events to occur simultaneously.

Notes

[11] The following two images are adopted from https://blogs.scientificamerican.com/roots-of-unity/the-subterfuge-of-epsilon-and-delta/

[12] Strictly speaking, we could introduce the chain rule and then show how the *product* and *quotient* are special cases. Also, the quotient rule is a special case of the product rule when we note that $\frac{1}{x} = x^{-1}$.

[13]'Infinitesimal' refers to a quantity that is extremely small and can be thought of as approaching zero. In calculus, infinitesimals are used to represent changes in variables that are so small that they can be approximated as being instantaneous. For example, if we are taking the total derivative of a function $f(x, y)$ with respect to x, then dx represents an infinitesimal change in the variable x, and dy represents an infinitesimal change in the variable y. The total derivative tells us how much the function changes when both x and y change by infinitesimally small amounts dx and dy, respectively. The use of infinitesimals is a key concept in calculus, and allows us to describe how functions change in response to small variations in their inputs.

[14]If all eigenvalues are positive, then the critical point is a local minimum. If all eigenvalues are negative, then the critical point is a local maximum. If the eigenvalues have both positive and negative values, then the critical point is a saddle point.

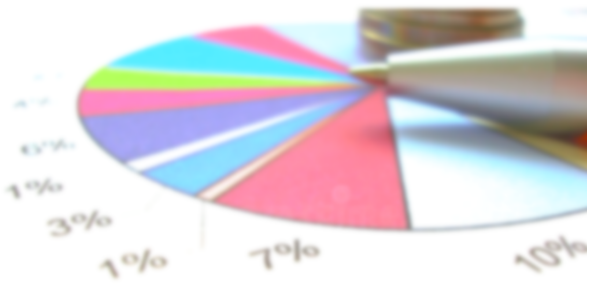
3.4 Further Reading

Here are a few useful resources on basic probability theory:

1. *Introduction to Probability* by Joseph K. Blitzstein and Jessica Hwang [6]. This textbook covers basic probability concepts and is suitable for beginners. It includes examples and exercises to help reinforce the material.
2. *Probability: Theory and Examples*, by Rick Durrett [11]. This textbook is a bit more advanced and is suitable for students with some background in calculus. It covers probability theory in depth, including measure theory and random processes. It also includes many examples and exercises.

Chapter 4

Statistics I: Descriptive Statistics



Data Scientists invest a lot in the pre-processing of data. This requires a good understanding of statistics. ***Statistics*** is a branch of mathematics that studies the collection, presentation, analysis, and interpretation of data. Statistical modeling lies at the heart of Data Science and Analysis!

Two main statistical methods are used in data analysis: descriptive statistics, and inferential statistics. ***Descriptive statistics*** is solely concerned with the properties of the observed data, and it does not rest on the assumption that the data may come from a larger population. In machine learning, the term 'inference' is sometimes used instead to mean 'make a prediction, by evaluating an already trained model'.

Inferential statistics, on the other hand, is the process of using data analysis to infer properties of an underlying distribution of probability. Inferential statistical analysis infers properties of a population, e.g., by testing hypotheses and deriving estimates. The observed data set is assumed to be sampled from a larger population, and the goal is to draw conclusions about that population based on the sample. In other words, inferential statistics is used to make inferences about a population based on a sample of data.

Data scientists use both descriptive and inferential statistics to gain insights from data. Descriptive statistics are used to summarize and describe the main features of a dataset, such as the mean, median, and standard deviation. These statistics can help data scientists understand the distribution of the data and identify any outliers or anomalies. Inferential statistics, on the other hand, are used to make predictions about a larger population based on a sample of data. This is often done by testing hypotheses and deriving estimates of population parameters such as the mean or proportion.

In this chapter, we dive deep into some of the most important topics in descriptive statistics. Chapter 6 will cover inferential statistics.

4.1 Measuring Central Tendency

We kick off our journey with the measures of location or central tendency. ***Central tendency*** is a descriptive statistical measure that represents the "center" or typical value of a data set. It is used to summarize the distribution of data and provide insight into its characteristics.

There are a few main measures of central tendency: mean (μ), median ($\tilde{x}$), and mode (M_o). Each of these measures provides a different way of describing the center of a data set, and they are each appropriate for different types of data.

4.1.1 Mean

The ***mean*** is the most commonly used measure of central tendency. It is calculated by adding up all the values in a data set and dividing by the number of values:

$$\mu = \frac{\sum_{i=1}^{n} x_i}{n} \tag{4.1}$$

For example, if we have the following data set of test scores: 75, 80, 85, 90, 95, the mean would be calculated as:

$$\mu = \frac{75 + 80 + 85 + 90 + 95}{5} = 85$$

This means that the average test score is 85.

4.1.2 Median

The ***median*** is the middle value in a data set when the values are arranged in order. If the data set has an odd number of values, the median is the middle value. If the data set has an even number of values, the median is the average of the two middle values:

$$\begin{cases} x_{(\frac{n+1}{2})} & n \text{ odd} \\ \frac{x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}}{2} & n \text{ even} \end{cases} \tag{4.2}$$

For example, if we have the following data set of test scores: 75, 80, 85, 90, 95, the median would be 85 because it is the middle value.

4.1.3 Mode

The ***mode*** is the value that appears most frequently in a data set. If there are multiple values that appear with the same frequency, the data set is said to have multiple modes:

$$M_o = \text{mode}(x) \tag{4.3}$$

For example, if we have the following data set of test scores: 75, 80, 85, 85, 90, 95, the mode would be 85 because it appears twice, which is more than any other value in the data set.

4.1.4 Percentiles and Quartiles

Percentiles and quantiles are measures of the relative position of a value in a dataset. ***Percentiles*** divide the dataset into 100 equal parts, while ***quantiles*** divide the dataset into any number of equal parts; in the specific case where this number is 4, we're dealing with *quantiles*. The pth percentile is the value below which p% of the data falls. The formula for the pth percentile is:

$$\text{Percentile}_p = \frac{p}{100}(n + 1) \tag{4.4}$$

where n is the number of data points in the dataset. The p^{th} quantile is defined in a similar way, but with the dataset divided into q equal parts:

$$\text{Quantile}_q = \frac{q}{100}(n + 1) \tag{4.5}$$

For example, let's consider the following dataset of exam scores:

$$\{65, 70, 72, 75, 80, 82, 85, 88, 90, 95\}$$

To find the 25th percentile, we need to identify the value below which 25% of the observations fall. To do this, we can arrange the data in order and count how many observations fall below the 25th percentile:

$$\{65, 70, 72, 75, 80, 82, 85, 88, 90, 95\}$$

There are 10 observations in the dataset, so the 25th percentile corresponds to the observation at position $0.25(10 + 1) = 2.75$. Since this is not a whole number, we can take the average of the 2nd and 3rd values to get the 25th percentile:

$$\text{25th percentile} = \frac{70 + 72}{2} = 71$$

To find the median or 50th percentile, we need to identify the value below which 50% of the observations fall. Since there are 10 observations in the dataset, the median corresponds to the observation at position $(10+1)/2 = 5.5$. Again, we can take the average of the 5th and 6th values to get the median:

$$\text{Median} = (80 + 82)/2 = 81$$

To find the 75th percentile, we need to identify the value below which 75% of the observations fall. Using the same approach as above, we can calculate the 75th percentile as:

$$\text{75th percentile} = (88 + 90)/2 = 89$$

Finally, we can use the concept of quantiles to divide the dataset into equal parts. For example, to find the quartiles (which divide the dataset into four equal parts), we can use the 25th, 50th, and 75th percentiles as the boundaries:

$$\begin{aligned} \text{First quartile (Q1)} &= 25th percentile = 71 \\ \text{Second quartile (Q2)} &= Median = 81 \\ \text{Third quartile (Q3)} &= 75th percentile = 89 \end{aligned}$$

4.2 Measuring Variability

Variability measures are statistical measures that describe the spread or dispersion of a dataset. They provide important information about the range and distribution of values in a dataset. Some common variability measures include range, interquartile range (IQR), variance, standard deviation, and standard error.

4.2.1 Range

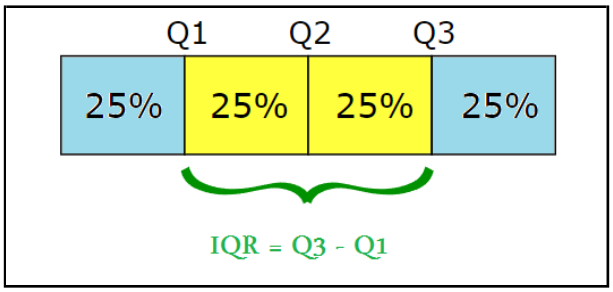
The ***range*** is a measure of the spread of a dataset and is calculated as the difference between the maximum and minimum values in the dataset. The formula for range is:

$$\text{Range} = \max(X) - \min(X) \tag{4.6}$$

The ***interquartile range (IQR)*** is a measure of the spread of a dataset that is based on the quartiles. It is defined as the difference between the third and first quartiles:

$$\text{IQR} = Q_3 - Q_1 \tag{4.7}$$

where Q_1 and Q_3 are the first and third quartiles, respectively. The second quartile, Q_2, is just the median.



Graphical Representations

In Python, we can conveniently represent the measures of central tendency and variability in graphical forms, using a range of libraries (such as Matplotlib and Seaborn) and their plots. One of these plots that's particularly relevant here is box plot.

A ***boxplot***, also known as a box and whisker plot, is a graphical representation of a dataset that shows the distribution of the data, including its central tendency, variability, and outliers. The boxplot consists of a rectangle (the box) with lines extending from it (the whiskers), with a point beyond the whiskers (the outlier) for any data points that are more than 1.5 times the interquartile range (IQR) away from the median.

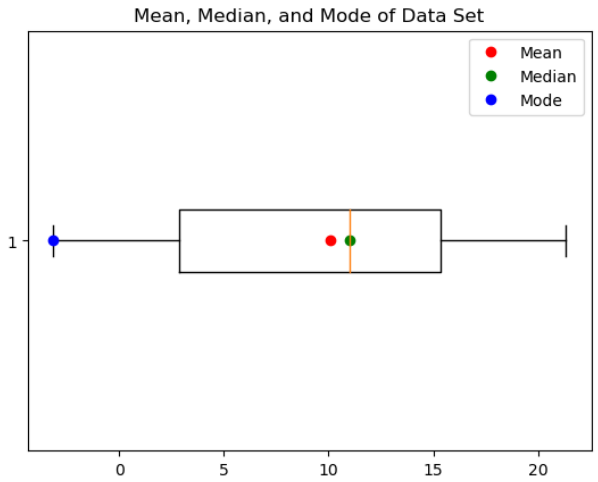
To create a boxplot, the data is first divided into quartiles: Q_1, Q_2 and Q_3. The box of the plot represents the IQR. The whiskers extend from the box to the minimum and maximum values that are not considered outliers.

The boxplot is useful for visualizing the spread and skewness of the data, as well as identifying any outliers. It can also be used to compare the distribution of multiple datasets side-by-side.

Example 4.2.1. Suppose we have the following random data:[15]

```
>> import numpy as np
>> data = np.random.normal(10, 6, 50)
>> print(data)
array([ 1.60693002, 18.31484875,  2.5071379 ,  6.98586967, 20.0492967 ,
        2.4698981 , 16.26439923, 10.81810438, 10.20782809,  2.19800013,
       16.97206225, 11.67773034, 16.09319014, 19.86203985, 13.882661  ,
       20.03807126, 21.06616342, 11.5667555 ,  0.91090276,  2.48467784,
       10.51151549,  1.45317773, 10.67013658, 15.76484335,  4.16688986,
        0.22269522, 14.76370054,  9.21668863,  7.38837841, 13.75533891,
       13.7320933 , -3.14906672, 13.76508068, 19.23606146,  7.15177284,
       11.86510987, 17.00140593, -2.91760274,  9.07984469,  2.55554955,
       11.20728568, 15.52108459, 13.51330191,  3.90400489, 11.22481934,
       21.32644116, 12.34874817,  1.24651264, 10.66011291,  2.33546663])
```

We can plot the box plot of the data and see its mean, median, and mode (more on box plots below):



The mean, the median and the mode of our data are shown in the plot. Their exact values are as follows:

```
Mean: 10.109
Median: 11.012
Mode: -3.149
```

4.2.2 Variance and Standard Deviation

Variance is a measure of the spread or dispersion of a dataset. It quantifies how much the individual values in a dataset vary from the mean value of the dataset. Variance is commonly used in statistics to analyze and compare datasets. The ***standard deviation*** is the square root of the variance.

Both standard deviation and variance are measures of dispersion that describe how spread out the data is from the mean. However, they have different properties and uses, and one may be more appropriate than the other depending on the situation.

One reason why standard deviation is often used instead of variance is that it has the same unit of measurement as the original data, while variance has units squared. This makes it easier to interpret the standard deviation in terms of the original data, and to compare the spread of data between different groups or variables that may have different units.

Another reason why standard deviation is often preferred is that it is more intuitive to understand and work with compared to variance. The standard deviation is in the same unit as the original data, and it represents the average distance of the data points from the mean. By contrast, the variance is in squared units, which can be harder to conceptualize.

However, variance also has some advantages over standard deviation in certain situations. For example, variance is a simpler and more fundamental measure of dispersion that is used in many statistical calculations and formulas, such as the calculation of confidence intervals and hypothesis tests. Additionally, the variance has some mathematical properties that make it easier to work with in some cases, such as when calculating the variance of a sum of independent random variables.

In summary, both standard deviation and variance are useful measures of dispersion, but they have different properties and uses, and one may be more appropriate than the other depending on the context and purpose of the analysis.

There are two types of variance (and accordingly, standard deviation): one for the entire *population*, and another for a *sample* of the population

Population variance is the variance of a complete population, which includes all possible values of the variable being measured. The population variance is denoted by σ^2 and is calculated as:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{4.8}$$

where N is the size of the population, x_i are the individual values in the population, μ is the population mean, and $(x_i - \mu)^2$ represents the squared deviation of each value from the population mean. Accordingly, the ***population standard deviation*** is determined using the following formula:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2} \tag{4.9}$$

Since it is usually impractical or impossible to measure an entire population, population variance is typically estimated using sample data.

Sample variance is the variance of a sample of data, which is a subset of the complete population. Sample variance is denoted by s^2 and is calculated as:

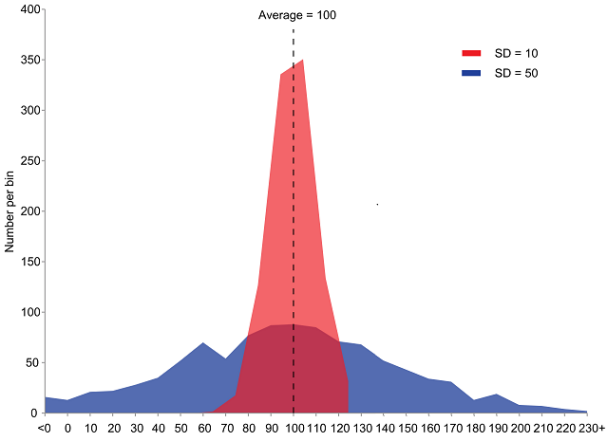
$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{4.10}$$

where n is the size of the sample, x_i are the individual values in the sample, $\bar{x}$ is the sample mean, and $(x_i - \bar{x})^2$ represents the squared deviation of each value from the sample mean. Accordingly, the ***population standard deviation*** is determined using the following formula:

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.11}$$

It is important to note that sample variance / standard deviation tends to underestimate the population variance / standard deviation due to the fact that the sample data is a smaller subset of the complete population. Therefore, a correction factor of $(n-1)$ is used instead of n to calculate sample variance, which increases the sample variance to account for this underestimation.

To see what insights can these measures bring to us, consider the following plot:[16]



This plot depicts samples from two populations with the same mean but different variances. The population with red shade has $\mu = 100$ and $s^2 = 100$ (standard deviation: 10) while the blue one has $\mu = 100$ and variance $s^2 = 2500$ (standard deviation: 50).

The fact that the two populations have the same mean but different variances indicate that they have different levels of variability in their data. The red population has a smaller variance and standard deviation, which means that the data points are more tightly clustered around the mean. On the other hand, the blue population has a larger variance and standard deviation, which means that the data points are more spread out from the mean.

This can have important implications in statistical analyses. For example, if you were to take random samples from these two populations, you would expect to see more variability in the blue population samples compared to the red population samples, even though the means of the two samples would be the same. Additionally, when performing hypothesis

tests or confidence intervals, the larger variance in the blue population could lead to wider intervals or larger p-values compared to the red population, even when the means are the same (more on hypothesis tests or confidence intervals later in the chapter).

4.2.3 Coefficient of Variation

The ***coefficient of variation (CV)*** is a statistical measure used to compare the variability of two or more datasets that have different units of measurement or different scales.

CV indicates how large the standard deviation is relative to the mean, and is defined as follows:

$$\text{CV} = (\frac{s}{\bar{x}} \times 100)\% \tag{4.12}$$

where $\bar{x}$ is the mean and s is standard deviation.

The coefficient of variation is useful for comparing the relative variability of datasets with different units or scales. A higher coefficient of variation indicates that a dataset has greater relative variability compared to its mean, while a lower coefficient of variation indicates that a dataset has less relative variability compared to its mean. The coefficient of variation is a dimensionless quantity, which makes it useful for comparing datasets with different units or scales.

For example, suppose we have two datasets: one measures the heights of a group of people in centimeters, and the other measures their weights in kilograms. The standard deviation of the height dataset will be in centimeters, while the standard deviation of the weight dataset will be in kilograms. Since the two datasets have different units, it may not be appropriate to compare the absolute values of their standard deviations. However, we can compare their coefficients of variation to see which dataset has relatively more variability.

More specifically, suppose we have a dataset of heights for a group of people, with a mean height of 170 centimeters and a standard deviation of 10 centimeters. We also have a dataset of weights for the same group of people, with a mean weight of 70 kilograms and a standard deviation of 5 kilograms.

To calculate the coefficient of variation for each dataset, we use the following formulas we proceed as follows:

- For the height dataset:

$$CV_h = \frac{10}{170} \times 100\% = 5.88$$

- For the weight dataset:

$$CV_w = \frac{5}{70} \times 100\% = 7.14$$

So, in this example, we can see that the coefficient of variation for the height dataset is lower than the coefficient of variation for the weight dataset. This indicates that the height dataset has relatively less variability compared to its mean, while the weight dataset has relatively more variability compared to its mean.

4.2.4 Standard Error

The ***standard error of the mean (SEM)*** is a measure of the variability of the sample mean from the true population mean. It is used when the data are continuous and the variable of interest is a mean or an average.

For example, the SEM might be used to estimate the precision of a sample mean blood pressure measurement as an estimate of the population mean blood pressure.

The formula is as follows:

$$\text{SEM} = \frac{s}{\sqrt{n}} \tag{4.13}$$

where s is the sample standard deviation, and n is the size of the sample.

On the other hand, the ***standard error of a proportion (SEP)*** is a measure of the variability of the sample proportion from the true population proportion. It is used when the data are categorical and the variable of interest is a proportion or a percentage.

For example, the standard error of a proportion might be used to estimate the precision of a sample proportion of patients with a certain disease as an estimate of the population proportion of patients with that disease.

The formula is as follows:

$$\text{SEP} = \sqrt{\frac{p(1-p)}{n}} \tag{4.14}$$

where p is the sample proportion, and n is the size of the sample.

In both cases, the standard error gives an estimate of how much the sample mean or proportion is likely to vary from the true population mean or proportion. The larger the standard error, the more variability there is in the estimate and the less precise it is likely to be.

Here is an example problem that involves computing variability measures and standard error:

Example 4.2.2. Suppose we are interested in estimating the average weight of a certain population of animals. We take a random sample of 20 animals and measure their weights in kilograms. The sample data is as follows (the matrix representation is just for the data to properly fit in the page – it doesn't really hinge on what matrices are):

$$\begin{bmatrix} 21 & 23 & 25 & 26 \\ 24 & 27 & 20 & 22 \\ 23 & 28 & 25 & 26 \\ 23 & 24 & 21 & 22 \\ 25 & 26 & 27 & 28 \end{bmatrix}$$

First, we calculate the sample mean:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{21 + 23 + 25 + \cdots + 27 + 28}{20} = 24.15$$

Then, we can calculate the sample variance:

$$s^2 = \frac{(21 - 24.15)^2 + \cdots + (28 - 24.15)^2}{19} = 4.89$$

Finally, we can calculate the sample standard deviation:

$$s = \sqrt{4.89} = 2.21$$

To calculate the standard error of the mean, we use the formula:

$$\text{SEM} = \frac{s}{\sqrt{n}}$$

where s is the sample standard deviation and n is the sample size. For our sample, we have:

$$\text{SEM} = \frac{2.21}{\sqrt{20}} = 0.494$$

Therefore, we can say that the average weight of the population is estimated to be 24.15 kg, with a standard error of 0.494 kg.

4.3 Measuring Association Between Two Variables

So far, we have examined ways to summarize data for one variable at a time. However, in many cases, a manager or decision maker may be interested in understanding the relationship between two variables. In this section, we will introduce covariance and correlation as measures to describe the relationship between two variables

4.3.1 Covariance

Covariance is a statistical measure that indicates the degree to which two variables are related. Specifically, it measures the extent to which changes in one variable are associated with changes in another variable. A positive covariance indicates that the variables tend to increase or decrease together, while a negative covariance indicates that one variable tends to increase while the other decreases.

Just like variance, covariance comes in two flavors. ***Population covariance*** measures the degree to which two variables in a population of data points vary together from their means:

$$\text{cov}(x, y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{4.15}$$

where μ_x and μ_y are the population means of x and y, and N is the population size.

Sample covariance measures the degree to which two variables in a set of *sample* data points vary together from their means:

$$\text{cov}(x, y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y}) \tag{4.16}$$

where $\bar{x}$ and $\bar{y}$ are the sample means of x and y, and n is the sample size.

Covariance and variance are related in that they both measure variability in a set of data, with covariance measuring the joint variability of two variables and variance measuring the variability of a single variable.

The variance of a variable can be seen as a special case of covariance, where the two variables are the same. Specifically, the variance of a variable is the covariance between that variable and itself. In other words, the variance of a variable is the degree to which it varies from its own mean. That is:

$$\text{var}(x) = \sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(x_i - \mu_x) = \text{cov}(x, x) \tag{4.17}$$

A real-world of where covariance is used is in the field of finance. Covariance is used to measure the relationship between the returns of two assets, such as stocks or mutual funds.

For example, let's consider two stocks, Stock A and Stock B. An investor might be interested in measuring the covariance between the returns of

these two stocks because it can provide insight into how they move together in the market.

Suppose we have a data set that contains the monthly returns of Stock A and Stock B over the past year. We can use this data to calculate the covariance between the two stocks. If the covariance is positive, it means that when Stock A has a high return in a given month, Stock B tends to have a high return as well. If the covariance is negative, it means that when Stock A has a high return in a given month, Stock B tends to have a low return, and vice versa.

The insight that covariance provides is that it helps us to understand the degree to which two assets are related in terms of their returns. If two assets have a high positive covariance, it means that they tend to move together in the market, which could indicate that they are highly correlated. On the other hand, if two assets have a negative covariance, it means that they tend to move in opposite directions in the market, which could indicate that they are negatively correlated.

By understanding the covariance between different assets, investors can make more informed decisions about how to allocate their investments in order to diversify their portfolios and reduce their overall risk.

4.3.2 Pearson Correlation Coefficient

Pearson correlation coefficient is a measure of the strength and direction of the linear relationship between two variables. However, it deserves a mention since we just went through covariance.

Pearson Correlation Coefficient is denoted by the symbol r and ranges from -1 to 1, where a value of -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.

Pearson correlation coefficient for a population can be calculated using the following formula:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \tag{4.18}$$

where $\text{cov}(X,Y)$ is the population covariance between X and Y, and σ_X and σ_Y are the population standard deviations of X and Y, respectively.

The formula for a *sample's* Pearson correlation coefficient is as follows:

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.19}$$

where n is the number of observations, x_i and y_i are the values of the two variables for the ith observation, $\bar{x}$ and $\bar{y}$ are the sample means of the two variables, and the numerator is the sample covariance of x and y, while the denominator is the product of their sample standard deviations.

The Pearson correlation coefficient is related to variance and covariance. Variance measures the spread or variability of a single variable, while covariance measures the joint variability between two variables. The Pearson correlation coefficient is calculated as the covariance between the two variables divided by the product of their standard deviations, which is a normalized version of covariance. In other words, this measures the strength and direction of the linear relationship between two variables after removing the effect of the variability of each variable.

Another way to think about this is that the Pearson correlation coefficient is a standardized version of covariance. Covariance is affected by the units of measurement of the variables, while the Pearson correlation coefficient is a unitless measure that is always between -1 and 1, making it easier to compare the strength of relationships between variables that may be measured in different units. Additionally, while covariance can be positive or negative, in any magnitude, the Pearson correlation coefficient is always between -1 and 1, which allows for easier interpretation and comparison of relationships between variables.

Example 4.3.1. We can use the Pearson correlation coefficient to investigate the relationship between age and income.

Let's say that researchers collect data from a sample of 100 people and record their ages and income. They want to know if there is a relationship between age and income. After calculating the Pearson correlation coefficient, they find that there is a positive correlation of 0.6, which means that as age increases, income tends to increase as well. This information can be used to make predictions about future income based on age or to inform policies related to retirement or income inequality.

More concretely, let's say that the mean age is 45 years old with a standard deviation of 10 years and the mean income is \$50,000 with a standard deviation of \$10,000.

The Pearson correlation coefficient between age and income is calculated to be:

$$r = \frac{\sum_{i=1}^{100}(age_i - 45)(income_i - 50,000)}{\sqrt{\sum_{i=1}^{100}(age_i - 45)^2}\sqrt{\sum_{i=1}^{100}(income_i - 50,000)^2}} = 0.6$$

This means that there is a positive correlation between age and income, where as age increases, income tends to increase as well.

4.4 Sampling and Bias

We finish this chapter by discussing two important notions in statistics: sampling and bias.

Sampling is the process of selecting a subset of individuals or data points from a larger population for the purpose of analysis. The goal of sampling is to obtain a representative sample that accurately reflects the characteristics of the larger population. In statistics, various tests are used to analyze data and draw inferences about the population based on the sample. These tests, such as t-tests and z-tests, rely on the assumption that the sample is representative of the population.

Bias is a systematic error or deviation from the true value that occurs in the sampling process. Bias can arise due to various reasons, such as the sampling method used, the sample size, and the characteristics of the population. A biased sample may not accurately reflect the population, leading to inaccurate inferences and conclusions.

To see sampling and bias in action, let's say a researcher is conducting a study to determine the average height of adult males in a particular city. They randomly select 100 men from a phone directory and measure their heights. However, they later realize that the phone directory only includes landline phone numbers, which may not be representative of the entire population. This introduces bias into the study, as it is possible that people who only have cell phones may have different heights on average than those who have landline phones.

In statistics tests, the presence of bias can affect the validity of the results. If a sample is biased, the statistical tests may produce results that are not representative of the population. For example, if a study on the effectiveness of a new drug only includes participants from a certain age group or demographic, the results may not be generalizable to the larger population.

To address bias, various sampling techniques are used, such as random sampling, stratified sampling, and cluster sampling. These techniques aim to reduce bias and obtain a representative sample. Additionally, statistical tests often incorporate methods to account for bias, such as adjusting for covariates or using weighted analyses.[17]

Notes

[15] I have produced these with the code snippet `np.random.normal(10, 6, 50)`. This generates an array of 50 random numbers drawn from a normal distribution with a mean of 10 and a standard deviation of 6 using NumPy's `random.normal()` function. Note that the mean and standard deviation of the data generated by `np.random.normal(10, 6, 50)` may not be exactly 10 due to the random nature of the sampling. However, as the sample size is increased, the sample mean and median will converge to the population mean of 10.

[16] This is taken from https://en.wikipedia.org/wiki/Variance.

[17] Sampling and bias are relevant to a variety of statistical tests, including t-tests, z-tests, chi-square tests, and f-tests (read Chapter 6 first, for a better understanding of this endnote. In the case of t-tests and z-tests, these tests are used to compare means between two groups or a sample and a known population. A representative and unbiased sample is necessary for these tests to produce accurate and generalizable results. If the sample is biased, the results may not accurately reflect the population, leading to incorrect conclusions.

Similarly, in chi-square tests, the data is often categorical and represents frequencies or proportions. A biased sample can lead to inaccurate inferences about the relationship between variables or groups. For example, if the sample is not representative of the population, the chi-square test may produce results that suggest a relationship where none exists.

F-tests are often used to compare variances or means between multiple groups or populations. Biased samples can affect the accuracy of these tests, as the sample may not accurately reflect the population variances or means.

To address this bias, the researcher could use a different sampling method, such as randomly selecting individuals from a list of registered voters, which is likely to be a more representative sample of the adult male population.

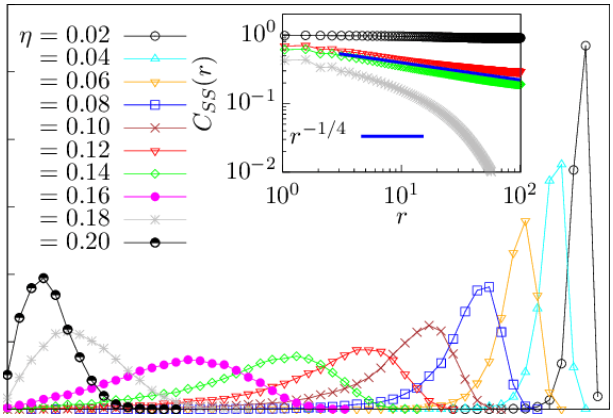
4.5 Further Reading

Here are a few useful resources on basic statistics:

1. *Statistics*, by David Freedman, Robert Pisani, and Roger Purves [29]. This textbook is widely used in introductory statistics courses and covers basic statistical concepts, such as probability, inference, and regression. It includes many examples and exercises to help reinforce the material.
2. *Elementary Statistics*, by Mario Triola [44]. This textbook is another popular choice for introductory statistics courses. It covers basic statistical concepts, such as descriptive statistics, probability, and hypothesis testing. It also includes many examples and exercises.
3. *The Practice of Statistics*, by Daren S. Starnes, Dan Yates, and David S. Moore [37]. This textbook is designed to teach students how to think statistically, rather than just memorize formulas. It covers basic statistical concepts, such as data collection, probability, and inference. It also includes many real-world examples and case studies.

Chapter 5

Probability Theory II: Probability Distributions



In Chapter 3 we covered the main basic concepts in the theory of probabilities. Chapter 4 gave us the equipment to understand the basics of statistics. Together these will come in handy in studying probability distributions.[18]

A ***probability distribution*** is a function that describes the likelihood of different outcomes or values of a random variable in a statistical experiment or process. A ***random variable*** is a numerical representation or description of the outcome of an experiment. Probability distribution functions provide ways of modeling uncertainty and predicting the likelihood of the outcomes.

The ***cumulative distribution function*** (***CDF***) of a random variable X is defined as the probability that X takes on a value less than or equal to x. It is denoted by F(x) and is defined mathematically as:[19]

$$F(x) = P(X \leq x) \tag{5.1}$$

There are two main types of probability distributions: continuous and discrete. A *discrete probability distribution* is used to describe the probability of a discrete or countable set of outcomes or values that a random variable can take on. A *continuous probability distribution* is used to describe the probability of a continuous or uncountable set of outcomes or values that a random variable can take on, such as all possible values within a range.

The choice of probability distribution to use depends on the nature of the data being modeled and the assumptions being made about the underlying process of generating the data.

5.1 Discrete Probability Distributions

A ***discrete probability*** distribution of a discrete random variable. A ***discrete random variable*** is a random variable that can assume either a finite number of values or an infinite sequence of values such as 0, 1, 2, 3, 4, ...

The ***Probability Mass Function*** (***PMF***) is a mathematical function that describes the probability distribution of a discrete random variable. Intuitively, the PMF assigns probabilities to each possible outcome of the random variable, and the sum of all of these probabilities must equal 1.

Formally, let X be a discrete random variable that can take on a finite or countably infinite number of values, and let $x_1, x_2, ..., x_n$ be the possible values that X can take on. Then, the PMF of X is defined as:

$$P(X = x_i) = p_i \tag{5.2}$$

where p_i is the probability that X takes on the value x_i. The PMF satisfies the following properties:

1. *Non-negativity:* $p_i \geq 0$ for all i.
2. *Normalization:* $\sum_{i=1}^{n} p_i = 1$.

The PMF can be used to calculate the probability of any event involving the random variable X. For example, the probability that X takes on a value less than or equal to a given value x_k can be calculated as:

$$P(X \leq x_k) = \sum_{i=1}^{k} P(X = x_i) = \sum_{i=1}^{k} p_i \tag{5.3}$$

Examples of discrete probability distributions include the Bernoulli Distribution, the binomial distribution and the Poisson distribution.

5.1.1 Bernoulli Distribution

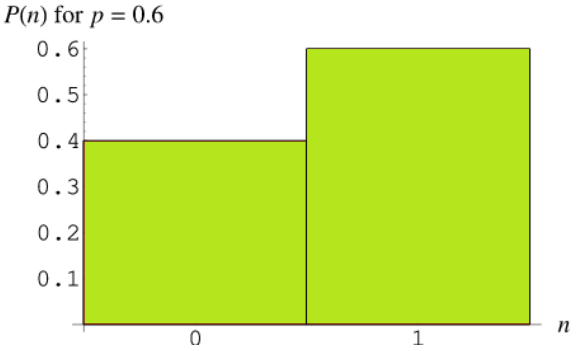
The ***Bernoulli distribution*** is a discrete probability distribution that models a random experiment with two possible outcomes, typically labeled as success (1) or failure (0).

Intuitively, the Bernoulli distribution can be thought of as a coin flip, where success corresponds to getting heads and failure corresponds to getting tails. More generally, it can represent any binary event, such as whether a customer makes a purchase or not.

Formally, let X be a random variable that takes the value 1 with probability p (i.e., success) and the value 0 with probability $1 - p$ (i.e., failure). We can express this as:

$$\begin{cases} P(X = 1) = p \\ P(X = 0) = 1 - p \end{cases} \tag{5.4}$$

where $0 \leq p \leq 1$.



The probability mass function of the Bernoulli distribution is:

$$f(x) = p^x (1 - p)^{1-x} \tag{5.5}$$

where $x = 0$ or 1.

The mean and variance of the Bernoulli distribution are:

$$\begin{cases} \mu = p \\ \sigma^2 = p(1 - p) \end{cases} \tag{5.6}$$

The cumulative distribution function (CDF) of the Bernoulli distribution is:

$$\begin{cases} 0 & \text{for } x < 0 \\ 1 - x & \text{for } 0 \leq x < 1 \\ 1 & \text{for } x \geq 1 \end{cases} \tag{5.7}$$

Example 5.1.1. An example of the Bernoulli distribution is a biased coin that has a 60% chance of landing heads (success) and a 40% chance of landing tails (failure). In this case, $p = 0.6$ and $1 - p = 0.4$, and the PMF of the Bernoulli distribution is:

$$f(x) = 0.6^x (0.4)^{1-x}$$

where $x = 0$ or 1.

5.1.2 Binomial Distribution

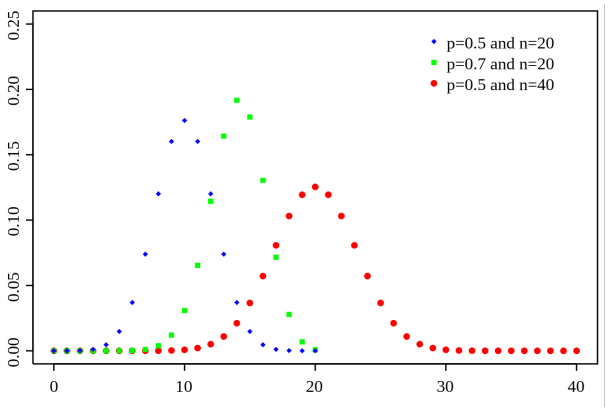
The ***binomial distribution*** is a discrete probability distribution that models the number of successes in a fixed number of independent and identically distributed Bernoulli trials.

Intuitively, the binomial distribution can be thought of as the distribution of the number of heads obtained in a fixed number of coin flips, where each flip is independent and has the same probability of heads. More generally, it can represent the number of successes in a fixed number of binary events, such as the number of customers who make a purchase out of a fixed number of website visitors.

Formally, let X be a random variable that represents the number of successes in n independent Bernoulli trials, each with the probability of success p. We can express this as:

$$X = X_1 + X_2 + \cdots + X_n \tag{5.8}$$

where each X_i is a Bernoulli random variable with parameter p, and X is the sum of the X_i's.[20]



The probability mass function of the binomial distribution is:

$$f(x) = \binom{n}{x} p^x (1-p)^{n-x} \tag{5.9}$$

where $x = 0, 1, \ldots, n$, and $\binom{n}{x} = \frac{n!}{x!(n-x)!}$ is the binomial coefficient.

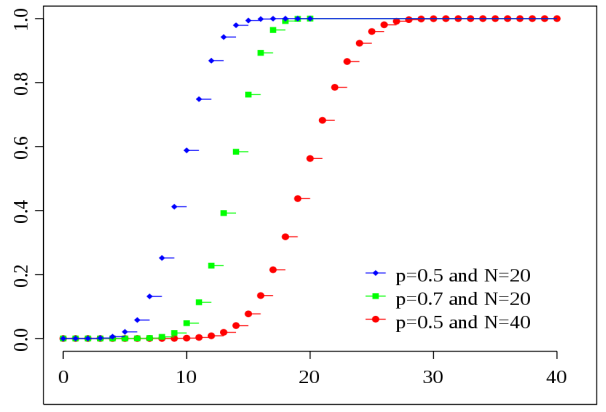
The mean and variance of the binomial distribution are:

$$\begin{cases} \mu = np \\ \sigma^2 = np(1-p) \end{cases} \tag{5.10}$$

The CDF of a binomial distribution can be written as:

$$F(k; n, p) = \sum_{i=0}^{\lfloor k \rfloor} \binom{n}{i} p^i (1-p)^{n-i} \tag{5.11}$$

where k is the number of successes, n is the total number of trials, and p is the probability of success in each trial. This formula calculates the probability of observing k or fewer successes in n trials with probability p of success in each trial.



Example 5.1.2. Let's build on top of our previous example. Suppose we're interested in the number of heads obtained in 10 flips, not just one, of a biased coin that has a 60% chance of landing heads (success). In this case, $n = 10, p = 0.6$, and the PMF of the binomial distribution is:

$$f(x) = \binom{10}{x} 0.6^x (0.4)^{10-x}$$

where $x = 0, 1, 2, ..., 10$.

5.1.3 Poisson Distribution

The ***Poisson distribution*** is a discrete probability distribution that models the probability of a certain number of events occurring in a fixed interval of time or space, given that these events occur independently and at a constant rate.

Intuitively, the Poisson distribution is often used to model rare events, such as the number of customers entering a store in an hour, the number of cars passing through an intersection in a minute, or the number of accidents at a particular intersection in a year. It assumes that the rate of occurrence of these events is constant over the interval of interest and that the events are independent of each other.

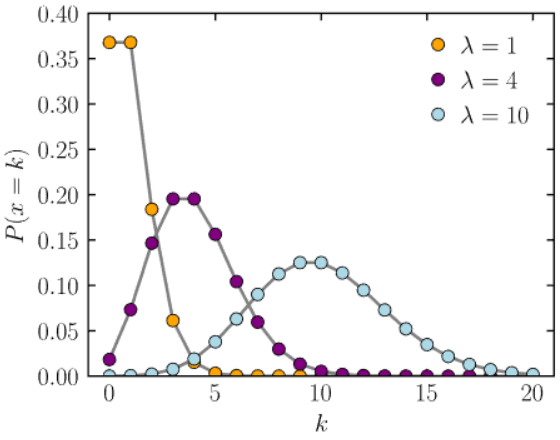
Rigorously, the Poisson distribution is defined by a single parameter, lambda (λ), which represents the expected number of events that occur in the interval of interest. The probability of observing k events in this interval is given by the following PMF:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!} \tag{5.12}$$

where e is the mathematical constant e, approximately equal to 2.71828, and $k!$ denotes the 'factorial' of k—that is, $k! = k \times (k-1) \times (k-2) \times ... \times 2 \times 1$.

The mean and the variance of the Poisson distribution function are as follows:

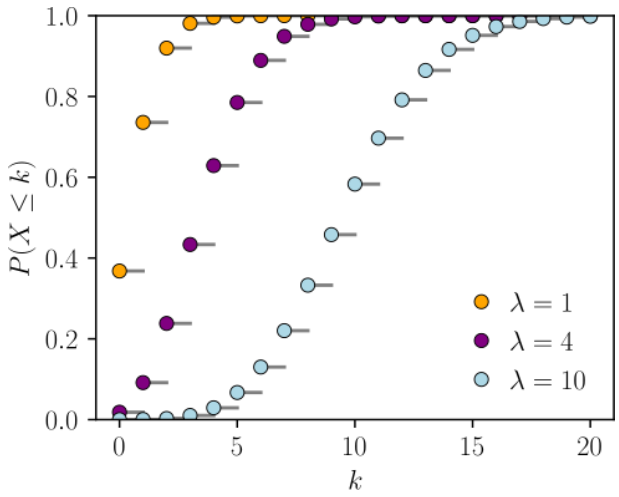
$$\begin{cases} \mu = \lambda \\ \sigma^2 = \lambda \end{cases} \tag{5.13}$$



The CDF of Poisson Distribution is as follows: The cumulative distribution function (CDF) of the Poisson distribution is:

$$F(k;\lambda) = \sum_{i=0}^{k} \frac{\lambda^i e^{-\lambda}}{i!} \tag{5.14}$$

where k is the number of events, and lambda is the mean number of events in the given time interval or region of space.[21]



Example 5.1.3. Suppose that on average, 5 cars pass through a particular intersection in a minute. We can model the number of cars that pass through this intersection in a minute using a Poisson distribution with parameter $\lambda = 5$. The probability of observing exactly 3 cars passing through the intersection in a given minute is:

$$P(X = 3) = \frac{5^3 e^{-5}}{3!} \approx 0.140$$

This means that there is a 14% chance of observing exactly 3 cars passing through the intersection in a minute, assuming that cars arrive independently and at a constant rate.

5.2 Continuous Probability Distributions

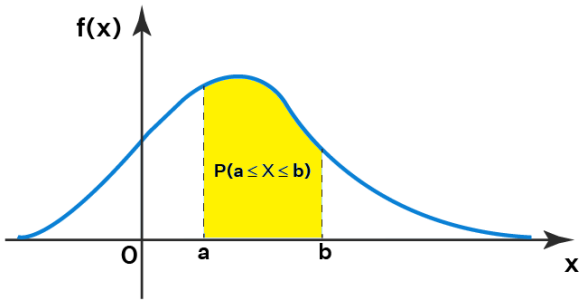
A ***continuous probability distribution*** is one in which the random variable can take on any value within a given range, and its probability distribution is described by a probability density function.

The ***probability density function*** (***PDF***) is a function that assigns probabilities to intervals of possible values rather than to individual values.

The PDF does not directly yield probabilities, but the area beneath the curve of the function over an interval determines the probability that the continuous variable takes on a value in that interval.[22]

Formally, let X be a continuous random variable, and let $f(x)$ be the PDF of X. Then, the probability that X takes on a value between two points a and b is given by the area under the PDF between a and b:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \tag{5.15}$$



The PDF satisfies the following properties:

1. *Non-negativity:* $f(x) \geq 0$ for all x.
2. *Normalization:* $\int_{-\infty}^{\infty} f(x)dx = 1$.

The PDF can be used to calculate the probability of any event involving the random variable X. For example, the probability that X takes on a value less than or equal to a given value x_k (also known as the c*umulative distribution function*, or *CDF*) can be calculated as:

$$P(X \leq x_k) = \int_{-\infty}^{x_k} f(x)dx \tag{5.16}$$

One example of a random variable that can be modeled using a PDF is the height of adult men, which is a continuous variable that can take on any value within a certain range. The PDF for this random variable might look like a normal distribution, with the peak of the curve representing the most common height value (see the subsection on normal distributions).

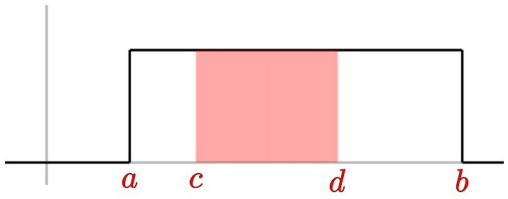
Below we discuss some well-known continuous probability distributions.

5.2.1 Uniform Distribution

The ***Uniform Distribution*** is a probability distribution where every value between a minimum and maximum value is equally likely, or 'uniform'. Intuitively, this means that if we were to plot the probability density function of the Uniform Distribution, we would get a rectangle with a constant height between the minimum and maximum values, and the area of the rectangle would be equal to 1.

Formally, let X be a continuous random variable with a uniform distribution over the interval $[a, b]$. Then, the probability density function of X is given by:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{5.17}$$



This means that the probability of X taking on a value between a and b is given by the area of the rectangle formed by the PDF:

$$P(a \leq X \leq b) = \int_a^b \frac{1}{b-a} dx = \frac{1}{b-a} \int_a^b dx = \frac{b-a}{b-a} = 1 \tag{5.18}$$

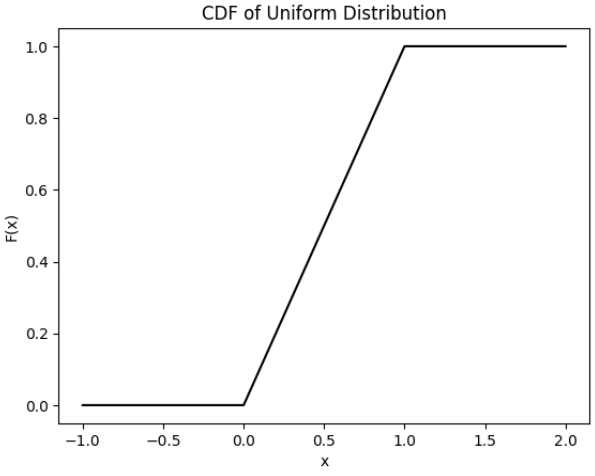
The mean and variance of X are given by:

$$\begin{cases} \mu = \frac{a+b}{2} \\ \sigma^2 = \frac{(b-a)^2}{12} \end{cases} \tag{5.19}$$

The cumulative distribution function (CDF) of a continuous uniform distribution with lower bound a and upper bound b is defined as:

$$F(x) = \begin{cases} 0 & x \leq a \\ \frac{x-a}{b-a} & a < x \leq b \\ 1 & x > b \end{cases} \tag{5.20}$$

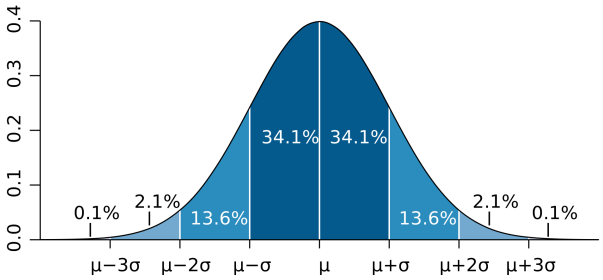
where x is a real number.



Example 5.2.1. An example of Uniform Distribution is the rolling of a fair die. Each number on the die has an equal chance of being rolled, and the minimum and maximum values are 1 and 6, respectively. Therefore, the probability of rolling any particular number is $\frac{1}{6}$, and the probability of rolling any number between 1 and 6 is 1.

5.2.2 Normal Distribution

Normal Distribution, also known as *Gaussian distribution*, is a continuous probability distribution that is commonly used to model natural phenomena such as the distribution of heights or weights in a population, IQ scores, and stock prices.[23] It is symmetric and bell-shaped, with the majority of data points falling near the mean (average) and fewer data points being farther away from the mean.[24]



Normal distribution has numerous applications in data science due to its unique properties. One of its primary uses is in inferential statistics. When we have a large sample size, it is often assumed that the underlying distribution of the data is normal. This assumption is crucial in hypothesis testing, as it allows us to use statistical tests that are specifically designed for normal distributions (more on hypothesis testing in the next chapter). Normal distribution is also used in machine learning, especially in regression models. Linear regression, for example, assumes that the errors of the model follow a normal distribution. When the errors are not normally distributed, the transformation of the variables or non-linear models might be used. These algorithms and their underlying mathematics will be discussed in Part II of the book.

The Gaussian distribution is characterized by two parameters: the mean, which determines the location of the peak, and the standard deviation, which controls the width of the distribution. The probability density function (PDF) of a normal distribution is given by:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.21}$$

where x is the random variable, μ is the mean, and σ is the standard deviation.

The ***standard normal model***, also known as the *standard normal distribution*, is a specific type of normal distribution that has a mean of zero and a standard deviation of one. It is often used in statistical analysis as a reference distribution to compare other normal distributions, or as a method for standardize data.

Here's the formula for standard normal distribution:

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2} \tag{5.22}$$

where z is a standardized random variable, obtained by the following formula:[25]

$$z = \frac{x - \mu}{\sigma} \tag{5.23}$$

The standard normal distribution is often used as a benchmark for other normal distributions, as any normal distribution can be transformed into the standard normal distribution using this formula:

Example 5.2.2. One example of using the normal distribution is in quality control. Let's say a factory produces bolts with an average length of 10 centimeters and a standard deviation of 0.1 centimeters. The lengths of the bolts can be assumed to follow a normal distribution with a mean of 10 centimeters and a standard deviation of 0.1 centimeters.

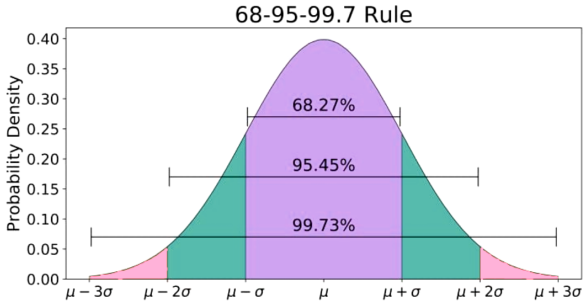
To ensure quality control, the factory may want to check a sample of bolts to make sure they meet certain specifications. For example, they may want to ensure that at least 95% of the bolts are between 9.8 and 10.2 centimeters in length.

To do this, they can use the formula for the normal distribution to calculate the probability of a bolt being within the specified range. They can then use this probability to determine the sample size needed to achieve a desired level of confidence in their quality control.

For example, if they want to be 95% confident that their sample will accurately reflect the quality of their entire production run, they can use the normal distribution to calculate the sample size needed to achieve this level of confidence. This can help them ensure that their customers receive high-quality products and minimize the risk of defects or recalls.

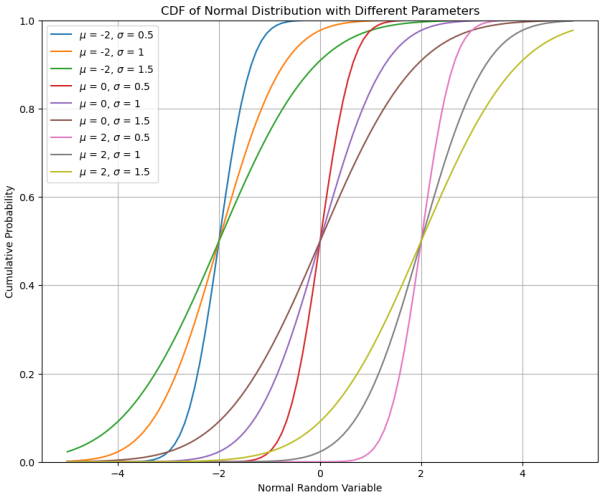
The ***empirical rule***, also known as the *68-95-99.7 rule*, is a statistical rule of thumb that applies to normal distributions. It states that:

1. Approximately 68% of the data falls within one standard deviation of the mean.
2. Approximately 95% of the data falls within two standard deviations of the mean.
3. Approximately 99.7% of the data falls within three standard deviations of the mean.



Example 5.2.3. If a normal distribution has a mean of 100 and a standard deviation of 10, the empirical rule tells us that approximately 68% of the data falls between 90 and 110, approximately 95% of the data falls between 80 and 120, and approximately 99.7% of the data falls between 70 and 130. The empirical rule is often used to quickly estimate the range of values that a normal distribution is likely to take.

The cumulative distribution function (CDF) of the standard normal distribution is a mathematical function that gives the probability that a standard normal random variable will take a value less than or equal to a certain value, z. This is a continuous and symmetric S-shaped curve that approaches 0 as z approaches negative infinity and approaches 1 as z approaches positive infinity.



The formula for the CDF of the standard normal distribution is:

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{\frac{-x^2}{2}} dx \tag{5.24}$$

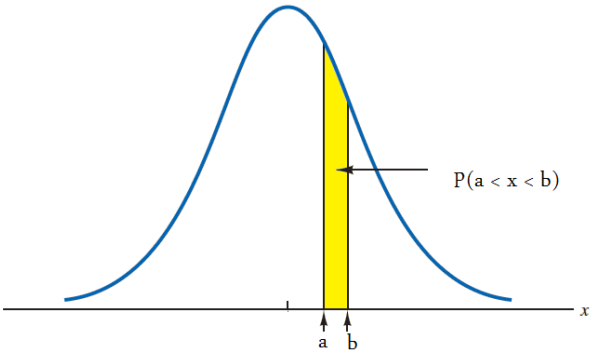
where e is the mathematical constant (approximately 2.718), π is the ratio of a circle's circumference to its diameter (approximately 3.141), and the integral is the area under the standard normal curve from negative infinity to z.[26]

The probability of a normal random variable X taking on a value between a and b can be calculated by finding the area under the probability density curve between the points a and b. This can be computed using the cumulative distribution function of the normal distribution, which gives the probability that a normal random variable is less than or equal to a given value.

The probability of X taking on a value between a and b is given by:

$$P(a < X < b) = P(X \leq b) - P(X \leq a) = \Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right) \tag{5.25}$$

where Φ is the standard normal cumulative distribution function, μ is the mean of the normal distribution, and σ is the standard deviation of the normal distribution.



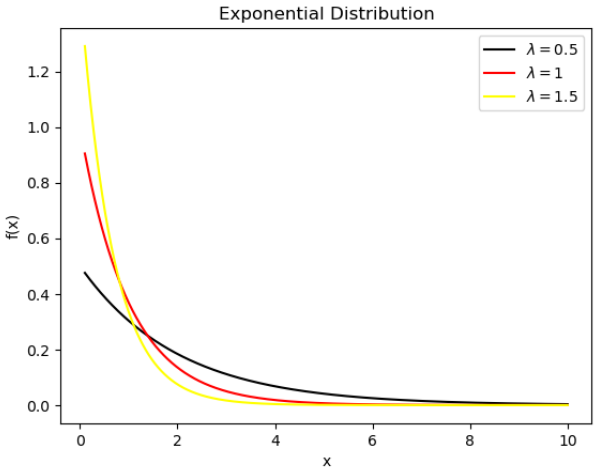
5.2.3 Exponential Distribution

The ***Exponential Distribution*** is a probability distribution that models the *time* it takes for an event to occur in a Poisson Process, where events occur at a constant rate and independently of one another. Intuitively, this means that the Exponential Distribution is often used to model the waiting time between successive events that occur at a fixed rate.

Formally, let X be a continuous random variable with an exponential distribution with parameter λ. Then, the probability density function of X is given by:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.26}$$

where $\lambda > 0$ is the rate parameter.



This means that the probability of X taking on a value between a and b is given by the area under the curve of the PDF between a and b:

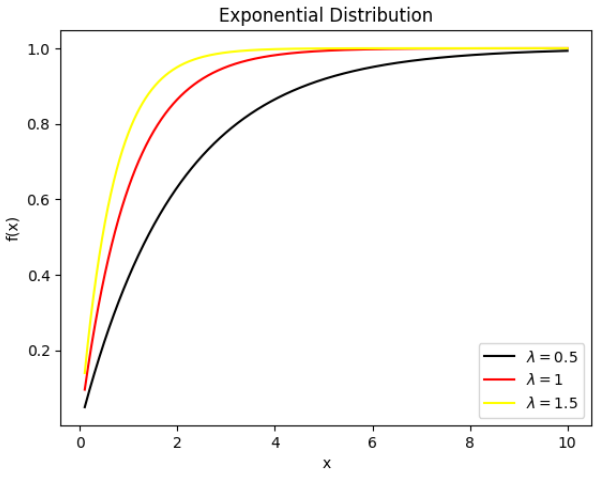
$$P(a \leq X \leq b) = \int_a^b \lambda e^{-\lambda x} dx = -e^{-\lambda x}\Big|_a^b = e^{-\lambda a} - e^{-\lambda b} \tag{5.27}$$

The mean and variance of X are given by:

$$\begin{cases} \mu = \frac{1}{\lambda} \\ \sigma^2 = \frac{1}{\lambda^2} \end{cases} \tag{5.28}$$

The cumulative distribution function (CDF) of the exponential distribution with parameter λ is:

$$F(x;\lambda) = \begin{cases} 1 - e^{-\lambda x}, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{5.29}$$



Example 5.2.4. An example of Exponential Distribution is the time between the arrivals of customers at a store, assuming that customers arrive at a constant rate of λ per unit time. In this case, X represents the waiting time between arrivals, and the parameter λ represents the arrival rate.

Suppose a store has an average arrival rate of 10 customers per hour. We can model the time between each customer's arrival using an exponential distribution with a rate parameter of $\lambda = 10$ customers per hour.

Using this information, we can calculate the probability of waiting a certain amount of time between two consecutive customers. For example, the probability of waiting more than 10 minutes between two consecutive customers can be calculated as:

$$P(X > 0.167) = e^{-10\times0.167} \approx 0.187$$

where X is the waiting time between customers in hours and 0.167 is the time in hours equivalent to 10 minutes.

This means that there is a 18.7% chance of waiting more than 10 minutes between two consecutive customers. Similarly, the probability of waiting more than 20 minutes can be calculated as:

$$P(X > 0.333) = e^{-20\times0.333} \approx 0.001$$

This means that there is an 0.1% chance of waiting more than 20 minutes between two consecutive customers.

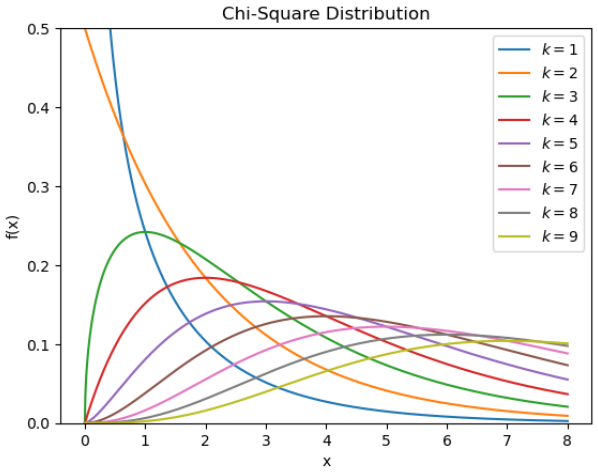
5.2.4 Chi-Square Distribution

The ***Chi-Square Distribution*** is a continuous probability distribution that arises when calculating the sum of the squares of independent standard normal random variables. Intuitively, this means that the Chi-Square Distribution is often used in hypothesis testing and confidence interval construction.

Formally, let $X_1, X_2, ..., X_k$ be independent standard normal random variables, and let $Y = X_1^2 + X_2^2 + ... + X_k^2$. Then, Y has a Chi-Square Distribution with k degrees of freedom, denoted by χ_k^2. The probability density function of the Chi-Square Distribution is given by:

$$f(x) = \frac{1}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})} x^{\frac{k}{2}-1} e^{-\frac{x}{2}} \tag{5.30}$$

where Γ is the gamma function.[27]



This means that the probability of Y taking on a value between a and b is given by the area under the curve of the PDF between a and b:

$$P(a \le Y \le b) = \int_a^b \frac{1}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})} x^{\frac{k}{2}-1} e^{-\frac{x}{2}} dx \tag{5.31}$$

The mean and variance of Y are given by:

$$\begin{cases} \mu = k \\ \sigma^2 = 2k \end{cases} \tag{5.32}$$

An example of the Chi-Square Distribution is in testing whether the variance of a population is equal to a specific value. In this case, we would use the Chi-Square Distribution to calculate the test statistic and p-value. We will discuss p-values in the next chapter.

We wrap up this section by introducing a canonical result in probability theory.

5.3 Central Limit Theorem

The ***Central Limit Theorem*** (***CLT***) is a fundamental concept in statistics that describes the behavior of the sum or average of a large number of independent and identically distributed random variables.

Independent and identically distributed (***iid***) random variables are a type of random variable that have a special relationship with each other: they're *independent*—meaning that the occurrence of one event does not affect the probability of the occurrence of the other event—and *identically distributed*—meaning that they have the same probability distribution, that is, they have the same mean, variance, and other statistical properties.

Now, Central Limit Theorem states that as the sample size n increases, the distribution of the sum

or average of the random variables becomes approximately normal, regardless of the underlying distribution of the individual random variables.

Intuitively, this means that if we repeatedly sample from any population with a finite mean and variance, and compute the average of each sample, the distribution of these sample averages will approach a normal distribution as the sample size increases.

We can express the CLT mathematically as this:

Theorem 5.3.1 (Central Limit Theorem)**.** Suppose that $X_1, X_2, ..., X_n$ are independent and identically distributed random variables with mean μ and variance σ^2. Let S_n denote their sum, and let:

$$Z_n = \frac{S_n - n\mu}{\sigma\sqrt{n}} \tag{5.33}$$

be the standardized variable.[28] Then, as n approaches infinity, the distribution of Z_n converges to a standard normal distribution, i.e.,

$$\lim_{n\to\infty} P(Z_n \leq z) = \Phi(z) \tag{5.34}$$

where $\Phi(z)$ is the cumulative distribution function of the standard normal distribution.

An example of the CLT in action would be rolling a fair six-sided die many times and computing the average value of the rolls for each sample. If we do this repeatedly and plot the distribution of these sample averages, we would expect to see a normal distribution emerge as the sample size increases.

Notes

[18]This image is taken from https://www.researchgate.net/publication/333161144_Re-entrant_motility_induced_phase_separation_in_nematically_aligning_active_polar_particles.

[19]For continuous random variables, the PDF is the derivative of the CDF. That is: $f(x) = \frac{d}{dx}F(x)$. For discrete random variables, the CDF is the sum of the probabilities of all values less than or equal to x. That is, the CDF F(x) of X is given by: $F(x) = \sum_i P(X = xi)$, for $xi \leq x$. The CDF provides a way to calculate the probability of a random variable taking on a value less than or equal to a given value. It also allows for the calculation of probabilities over a range of values by taking the difference between the CDF values at the two endpoints of the range.

[20]The next two images are taken from https://en.wikipedia.org/wiki/Binomial_distribution

[21]The image above and the one below are taken from https://en.wikipedia.org/wiki/Poisson_distribution

[22]Due to the fact that the area beneath the pdf at any specific point is zero, one consequence of the probability definition for continuous random variables is that the probability of any individual value of the variable is zero. Note that this is not a shortcoming of PDFs. This is a fundamental property of continuous random variables, and it arises due to the fact that the probability of a single point in a continuous distribution is infinitesimally small. This is because there are an infinite number of possible values that a continuous random variable can take on, and the probability of any individual value is effectively zero. Instead, we calculate the probability that the variable takes on a range of values, which is represented by the area under the PDF curve over that range.

[23]Here after I will use both of these terms interchangeably.

[24]The image below is taken from https://en.wikipedia.org/wiki/Probability_theory

[25]The 'z' here comes from the 'z-score'. See the next chapter for a detailed discussion of z-scores.

[26]In general, the CDF of the standard normal distribution is widely used in probability theory and statistics, as it allows us to calculate probabilities associated with normal distributions. For example, if we want to find the probability that a standard normal random variable is less than or equal to 1.96, we can use the CDF of the standard normal distribution to find that $\Phi(1.96) = 0.975$. This means that the probability of a standard normal random variable is less than or equal to 1.96 is 0.975 or 97.5%.

[27]The gamma function, denoted by $\Gamma(x)$, is a mathematical function that generalizes the factorial function. It is defined for all positive real numbers x and satisfies the recurrence relation $\Gamma(x+1) = x\,\Gamma(x)$ and the initial condition $\Gamma(1) = 1$. The gamma function is widely used in many areas of mathematics and science, including probability theory, statistics, and physics. In the formula for the chi-square probability density function, the gamma function appears in the denominator as $\Gamma(\frac{k}{2})$. This term is necessary to ensure that the function integrates to 1 over its support. In particular, for k an even positive integer, $\Gamma(\frac{k}{2}) = \frac{(k/2-1)!}{2^{k/2-1}}$, which can be seen by using the formula $\Gamma(x+1) = x\Gamma(x)$ repeatedly.

[28]Standardizing the variable makes it have a standard normal distribution with a mean of 0 and a variance of 1. This transformation is also known as the z-score transformation. The resulting standardized variable is denoted by the symbol Z. Technically, we can express the central limit theorem without using standardized variables, but using the standardized variable is a common way to state the theorem because it simplifies the calculations and makes it easier to apply the theorem in practice.

5.4 Further Reading

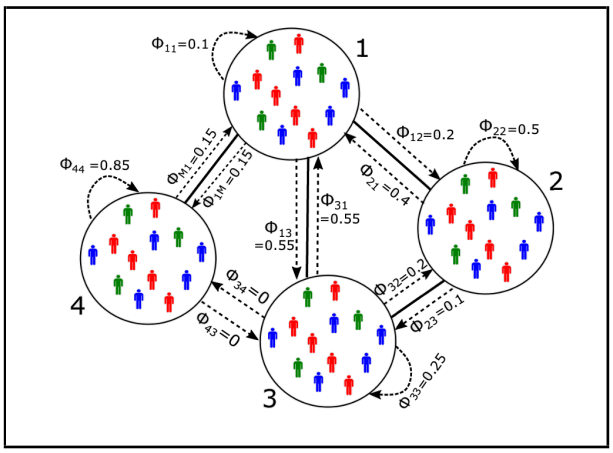
Here are a few useful resources on probability theory and distributions:

1. *Probability and Random Processes*, by Grimmett and Stirzaker [13]. This textbook provides an introduction to probability theory and stochastic processes, with an emphasis on the applications of these concepts to real-world problems. It covers topics such as discrete and continuous random variables, probability distributions, Markov chains, and Poisson processes. It is an ideal resource for students and practitioners in data science and related fields.
2. *Introduction to Probability Models*, by Sheldon Ross [32]. This textbook provides an introduction to probability theory and its applications, with a focus on modeling and simula-

tion. It covers topics such as discrete and continuous random variables, probability distributions, Markov chains, and Monte Carlo methods. It is an excellent resource for students and practitioners in data science and related fields.

Chapter 6

Statistics II: Inferential Statistics



In Chapter 4 we explored descriptive statistics. We now attend to inferential statistics.[29] Inferential statistics involves the use of probability theory and statistical inference techniques to estimate population parameters, test hypotheses, and make predictions about future outcomes. It involves analyzing data to determine whether any observed differences or relationships are statistically significant, meaning they are unlikely to have occurred by chance.

In this section, we explore test statistics and statistical significance which constitute the heart of inferential statistics.

6.1 Test Statistics and Statistical Significance

Test statistics and statistical hypothesis help researchers make decisions about the data they collect. A ***test statistic*** is a value derived from the sample data used to evaluate the 'null hypothesis'.

The ***null hypothesis***, denoted by H_0, is the statement that suggests there's no significant difference or relationship between variables; the ***alternative hypothesis***, H_a, is the negation or rejection of the null hypothesis. Researchers typically use test statistics to calculate the probability that the observed results occurred by chance alone, and based on the results, they accept or reject the null hypothesis.

The null and alternative hypotheses present opposing claims about the population, where only one of them can be true at a time. Ideally, the testing process should result in accepting H_0 when it's true, and rejecting it when H_a is true. However, due to the fact that hypothesis testing relies on sample data, it is possible to make errors.

There are two types of errors that can occur in hypothesis testing: Type I and Type II errors.

Type I error occurs when a null hypothesis (H_0) is rejected even though it is actually true. It is also called a *false positive*.

An example of a Type I error could be a drug company claiming that their new medication is effective at reducing the symptoms of a certain disease when it actually has no effect. Another example is a man taking a pregnancy test and the result coming back positive.

Type II error occurs when a null hypothesis is not rejected even though it is actually false. It is also called a *false negative*.

An example of a Type II error could be a medical researcher failing to detect a significant difference in blood pressure between two groups of patients when one group was given a new medication and the other was given a placebo, when in fact the medication did have an effect on blood pressure. Another example is a patient with STD taking an STD test and getting a negative result.

The following table, also known as the ***confusion materix***, can help remembering these, using examples:[30]

	Null Hypothesis is TRUE	Null Hypothesis is FALSE
Reject null hypothesis	Type I Error (False positive) (Pregnant man)	Correct Outcome! (True positive)
Fail to reject null hypothesis	Correct Outcome! (True negative)	Type II Error (False negative) (STD patient gets negative STD test)

Finally, let's define critical values. ***Critical values*** are the values of a test statistic that separate the rejection region from the acceptance region in a hypothesis test. The *rejection region* is the range of sample statistics that are unlikely to be obtained by chance, assuming the null hypothesis is true. The *acceptance region* is the range of sample statistics that are likely to be obtained by chance, assuming the null hypothesis is true.

Critical values are determined based on the chosen level of significance and the degrees of freedom associated with the test statistic.

The ***level of significance***, also known as α (alpha), is the probability of rejecting the null hypothesis when it is actually true. In other words, it is the probability of making a Type I error when the null hypothesis is true.

The ***degree of freedom*** (df) represents the number of independent observations used to calculate a test statistic. It is important because it influences the distribution of the test statistic and can affect the accuracy of the test results.

The degree of freedom is calculated as the sample size minus the number of restrictions, which are typically the parameters estimated as intermediate steps in computing the statistic.

For a sample of size n with r restrictions, the degree of freedom is calculated as:

$$df = n - r \tag{6.1}$$

For instance, if we have a sample of n observations and we estimate the mean of the population using one parameter, then the degree of freedom is $n - 1$. The reason for subtracting 1 from the total number of observations is that one observation is used to estimate the sample mean, and the remaining $n - 1$ observations are free to vary. This is because the sample mean is a fixed value that is calculated from the sample, so once it is known, there is only $n - 1$ observations left that can vary.

A higher degree of freedom leads to a more normal-shaped distribution and a more accurate estimate of the population parameters, while a lower degree of freedom leads to more variability in the estimates and a more skewed distribution.

It's important to note that the degrees of freedom cannot be negative, which means that the number of parameters you estimate cannot exceed the sample size. Essentially, the larger the degrees of freedom, the more reliable the statistical inference.

In what follows, we will discuss some common test statistics methods in detail.

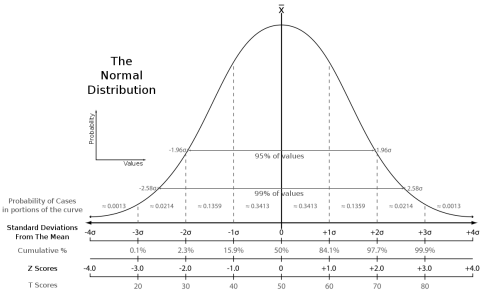
6.1.1 Z-Tests

Z-tests or *standardized tests* are statistical tests used to determine whether the mean of a single sample is statistically different from a known population mean. The test is based on the normal distribution and calculates the 'z-score', which measures the number of standard deviations a data point is from the mean.

Intuitively, the z-test can be thought of as a way to compare two groups and see if they are truly different from each other (i.e. if their difference is 'statistically significant') or if any observed difference is due to chance or some random variation. It helps to answer questions such as "Is the mean height of men different from the mean height of women?" or "Is there a significant difference in the mean income of people who graduated from college versus those who did not?"

The null hypothesis in a z-test is the statement that there is no significant difference between a population parameter and a sample statistic. In other words, it assumes that any observed difference between the sample and population is due to chance or random variation, rather than a true effect.

The critical value used in a z-test is determined by the level of significance, the sample size, and the type of test (one-tailed or two-tailed). In most cases, it is set at 0.05, which means that we are willing to accept a 5% chance of rejecting the null hypothesis when it is actually true.[31]

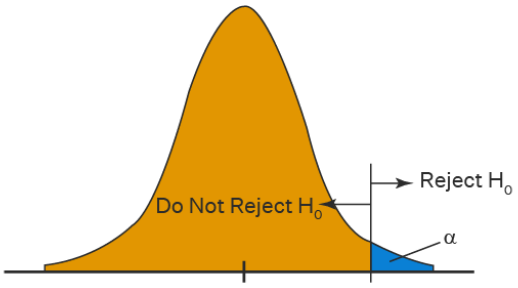


A ***one-tailed test*** is used when we are interested in testing a directional hypothesis, which means we

have a specific prediction about the direction of the effect. For example, we might predict that a new medication will improve cognitive function in older adults. A one-tailed test would be appropriate in this case because we are only interested in detecting an improvement, not a decline, in cognitive function.

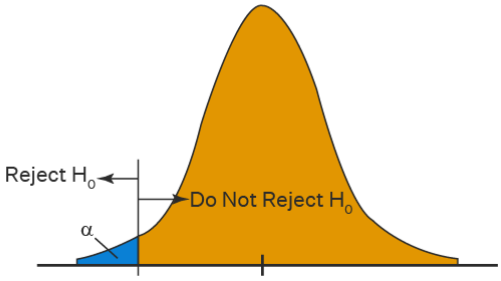
One-tailed tests can be either upper-tailed or lower-tailed. An ***upper-tailed*** z-test is a statistical hypothesis test where the null hypothesis is rejected if the test statistic is greater than the critical value, and the alternative hypothesis is that the population parameter is greater than the null hypothesis value.[32]

Right Tail Hypothesis Testing



In a ***lower-tailed*** z-test, on the other hand, the null hypothesis is rejected if the test statistic is smaller than the critical value, and the alternative hypothesis is that the population parameter is less than the null hypothesis value.

Left Tail Hypothesis Testing



The critical value used in a one-tailed z-test with a level of significance of 0.05 is 1.645. This value corresponds to the 5th percentile of the standard normal distribution. In other words, if we assume the null hypothesis is true, and we calculate a z-score that falls in the upper 5% of the distribution, we will reject the null hypothesis.

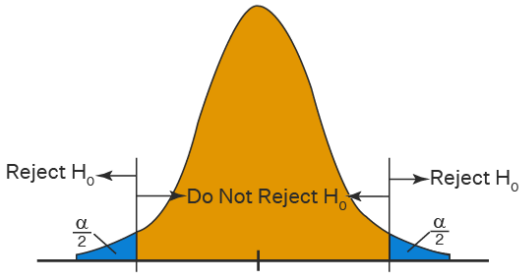
More generally, the critical value for a one-tailed z-test with a significance level of α can be calculated using the formula:

$$z_\alpha = \pm\Phi^{-1}(1-\alpha) \tag{6.2}$$

where Φ^{-1} is the inverse of the cumulative distribution function of the standard normal distribution. The sign of + or − depends on the direction of the test: + for upper-tailed and − for lower-tailed.

On the other hand, a ***two-tailed test*** is used when we are interested in testing a non-directional hypothesis, which means we have no specific prediction about the direction of the effect. For example, we might predict that a new medication will have a different effect on cognitive function compared to a placebo. A two-tailed test would be appropriate in this case because we are interested in detecting any difference, regardless of the direction.

Two Tail Hypothesis Testing



The critical value used in a two-tailed z-test with a level of significance of 0.05 is 1.96. This value corresponds to the 2.5th and 97.5th percentiles of the standard normal distribution. In other words, if we assume the null hypothesis is true, and we calculate a z-score that falls in the upper 2.5% or lower 2.5% of the distribution, we will reject the null hypothesis.

More generally, the critical value for a two-tailed z-test with a significance level of α can be calculated using the formula

$$z_{\frac{\alpha}{2}} = \pm\Phi^{-1}(1-\frac{\alpha}{2}) \tag{6.3}$$

The formula for calculating the z-score is the same as one-tailed and two-tailed z-tests (see first the formula below). However, as we noticed, the critical values used to determine the rejection region are different for one-tailed and two-tailed tests. In a two-tailed test, the rejection region is split between the upper and lower tails of the distribution, while in a one-tailed test, the rejection region is located entirely in one tail of the distribution.

One-Sample Z-tests

Often, and particularly in the paragraphs above, the term 'z-test' is used to refer specifically to the one-sample z-test. The ***one-sample z-test*** is used to

determine whether the mean of a single sample is statistically different from a known population mean.

The formula for a one-sample z-score is:

$$z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}} \tag{6.4}$$

where $\bar{x}$ is the sample mean, μ is the population mean, σ is the population standard deviation, and n is the sample size.

In a one-sample z-test, the null hypothesis is that the mean of a single population is equal to a specified value, and the alternative hypothesis is that they aren't equal to that value.

Note that in a z-test, there is no degree of freedom because here the population standard deviation is known. In other words, the sample size is large enough such that the sample standard deviation can be used as an estimate of the population standard deviation. (This is in contrast to a t-test, where the population standard deviation is unknown and must be estimated from the sample, leading to a loss of degrees of freedom. See the next section for more on t-tests.)

Example 6.1.1. The height of men is often assumed to follow a normal distribution, with a mean height and a standard deviation that varies depending on the population being considered. For example, suppose the mean height for adult men is approximately $5'75''$ (175 cm), and the standard deviation is approximately 2.8 inches (7.1 cm).

Assuming a normal distribution, we can use the probabilities associated with the distribution to make predictions about the heights of men. For example, we can use the normal distribution to answer questions such as:

- What is the probability that a randomly selected man is taller than 6 feet (183 cm)?
- What is the probability that a group of 100 randomly selected men has an average height of at least 6'2" (188 cm)?

To answer these questions, we can use the normal distribution function with the appropriate mean and standard deviation values. We can then use the cumulative distribution function (CDF) of the normal distribution to find the probabilities associated with the questions.

For example, to answer the first question, we can standardize the height of 6 feet using the formula:

$$z = \frac{\bar{x} - \mu}{\sigma} = \frac{6 - 5.75}{2.8} = 0.089$$

where $\bar{x}$ is the height of interest, μ is the mean height, and σ is the standard deviation. We can then find the probability that a standard normal variable is greater than 0.089 using a normal distribution table, a statistical software or a programing language like Python.[33] This gives us a probability of approximately 0.4641 or 46.41%, which means that there is a 46.41% chance that a randomly selected man is taller than 6 feet.

For the second question, we can use the central limit theorem, which states that the distribution of the sample mean for a large sample size is approximately normal, regardless of the underlying distribution of the population.

Assuming a normal distribution for the heights, we can calculate the probability that the sample mean height is at least 6'2" (188 cm) using the formula:

$$z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}} = \frac{6.2 - 5.75}{\frac{2.8}{\sqrt{100}}} = 1.79$$

where $\bar{x}$ is the sample mean height, n is the sample size, and the other variables are as before. We can then find the probability that a standard normal variable is greater than 1.79 using a normal distribution table or a statistical software, or just outright use the cumulative distribution function (CDF) of the standard normal distribution.[34] This gives us a probability of approximately 0.0367 or 3.67%, which means that there is a 3.67% chance that a group of 100 randomly selected men has an average height of at least 6'2" (188 cm).

Two-Sample Z-tests

What was discussed above mainly concerned with one-sample z-tests, which determine whether the mean of a single sample is statistically different from a known population mean. A ***two-sample z-test*** is a statistical test used to compare the means of two independent samples to determine if they are statistically different from each other. It is a hypothesis test that can be one-tailed or two-tailed depending on the directionality of the research question or hypothesis.

The two-sample z-test is used when the population standard deviations are known, or the sample sizes are large, and the null hypothesis is that the means of the two populations are equal, while the alternative hypothesis is that they are not equal.

Two-sample z-tests are performed using the following formula:

$$z = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}} \tag{6.5}$$

where $\bar{x}_1$ and $\bar{x}_2$ are the means of the two samples μ_1 and μ_2 are the population means of the two samples

σ_1 and σ_2 are the population standard deviations of the two samples, and n_1 and n_2 are the sample sizes of the two samples.

This formula is used when we want to compare the means of two independent samples to determine if they are statistically different from each other. (So, the null hypothesis here is that the means of two populations are equal, and the alternative hypothesis is that the means are different.) The two samples can be from the same population at different times or from different populations altogether.

Example 6.1.2. Suppose we want to test whether there is a significant difference in the mean weight of male and female college students. We collect a sample of 50 male college students and find that their average weight is 175 pounds with a standard deviation of 10 pounds. We also collect a sample of 50 female college students and find that their average weight is 160 pounds with a standard deviation of 8 pounds.

To test whether the difference in means is statistically significant using the z-test, we can follow these steps: First, we state the null and alternative hypotheses. The null hypothesis (H0) says that there is no significant difference between the mean weight of male and female college students; the alternative hypothesis (Ha) says otherwise.

We then determine the significance level (α), which is usually set at 0.05, and calculate the test statistic, which is the z-score:

$$z = \frac{(175 - 160) - 0}{\sqrt{\frac{10^2}{50} + \frac{8^2}{50}}} = 3.52$$

Now, we determine the critical value for the z-test at the chosen alpha level. Since our alternative hypothesis is two-tailed, we need to find the critical values for a two-tailed test. Using the relevant formula, we find that the critical values for a two-tailed z-test at alpha = 0.05 are -1.96 and +1.96.[35]

Finally, we compare the test statistic with the critical value. Since our z-score of 3.52 is greater than the critical value of +1.96, we can reject the null hypothesis and conclude that there *is* a significant difference between the mean weight of male and female college students at the alpha = 0.05 level.

6.1.2 T-Tests

A ***t-test*** is another type of statistical test used to compare the means of two groups and determine whether they are significantly different from each other. It is a more flexible test than the z-test because it can be used when the population standard deviation is not known, and the sample size is small.

Just like z-tests, there are two main types of t-tests: the one-sample t-test and the two-sample t-test. The ***one-sample t-test*** is used to compare the mean of a single sample to a known population mean when the population standard deviation is unknown. The test statistic for a one-sample t-test is calculated as:

$$t = \frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}} \tag{6.6}$$

where $\bar{x}$ is the sample mean, μ is the population mean, s is the sample standard deviation and n is the sample size.

The ***two-sample t-test*** is used to compare the means of two independent samples. The test statistic for a two-sample t-test is calculated as:

$$t = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{6.7}$$

where $\bar{x}_1$ and $\bar{x}_2$ are the means of the two samples, s_1 and s_2 are the standard deviations of the two samples, and n_1 and n_2 are the sample sizes of the two samples.

The null hypothesis for both types of t-tests is that the means of the samples are equal, while the alternative hypothesis is that they are not equal.

We could use the ***pooled standard deviation*** in calculating the t-score for two-sample t-tests when the variances of the two populations are equal, or when assuming so seems helpful (e.g., we don't have prior knowledge of the variance of the populations). If this assumption is reasonable, using the pooled standard deviation can increase the precision of the t-test and improve its power.[36]

The formula for the pooled standard deviation is:

$$s_p = \sqrt{\frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}} \tag{6.8}$$

where s_1 and s_2 are the standard deviations of the two samples, and n_1 and n_2 are the sample sizes.

The formula for the t-score using the pooled standard deviation is:

$$t = \frac{(\bar{x}_1 - \bar{x}_2) - (\mu_1 - \mu_2)}{\sqrt{\frac{s_p^2}{n_1} + \frac{s_p^2}{n_2}}} \tag{6.9}$$

where $\bar{x}_1$ and $\bar{x}_2$ are the means of the two samples, μ_1 and μ_2 are the population means (which we assume to be unknown), and s_p is the pooled standard deviation.

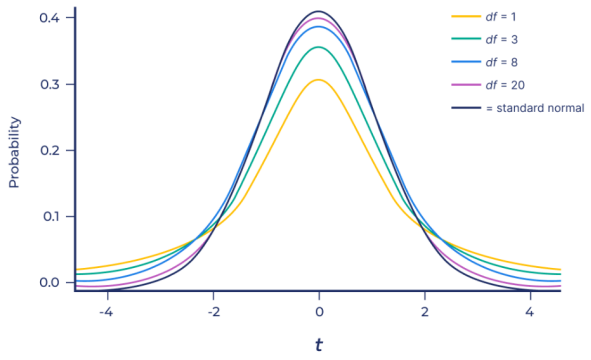
In t-tests, the degrees of freedom represent the number of observations in a sample that are free to vary after estimating one or more parameters from the sample. For example, in a one-sample t-test, the sample mean is used to estimate the population mean, so one degree of freedom is lost. In a two-sample t-test, both sample means are used to estimate the population means, so two degrees of freedom are lost.

Here are the formulas for the degrees of freedom of a t-test:

Test	Degrees of Freedom
One-sample t-test	$df = n - 1$
Two-samples t-test	$df = n_1 + n_2 - 2$

where n_1 is the sample size of group 1 and n_2 is the sample size of group 2.

The degrees of freedom are important because they affect the shape of the t-distribution: as the degrees of freedom increase, the t-distribution becomes more similar to the standard normal distribution. On the other hand, if the degrees of freedom are small, the t-distribution is wider and flatter.[37]



This relationship between degrees of freedom and spread intuitively makes sense, as well, because larger sample sizes provide more certainty in the estimate. To conceptualize this, consider repeatedly sampling from the population and calculating the t-statistic each time. As the sample size increases, the variability of the test statistic between samples decreases. This reduced variability reflects the increasing certainty of the estimate, resulting in a narrower distribution with less spread.

As for critical values in t-tests, the formulas for calculating critical values for one-sample and two-sample t-tests are similar, but with different degrees of freedom and significance levels.

For a one-sample t-test, the critical value is found using the t-distribution with $n-1$ degrees of freedom, where n is the sample size. The formula for finding the critical value is:

$$t_{\frac{\alpha}{2},n-1} = \pm t_{1-\frac{\alpha}{2},n-1} \tag{6.10}$$

where $t_{\frac{\alpha}{2},n-1}$ is the critical value, $t_{1-\frac{\alpha}{2},n-1}$ is the t-value that corresponds to the upper (or lower) tail area of the t-distribution with $n-1$ degrees of freedom and a significance level of $\frac{\alpha}{2}$, and $\pm$ indicates whether the critical value is positive or negative, depending on whether it is a one-tailed or two-tailed test.

For a two-sample t-test with equal variances, the critical value is found using the t-distribution with $n_1 + n_2 - 2$ degrees of freedom, where n_1 and n_2 are the sample sizes. The formula for finding the critical value is:

$$t_{\frac{\alpha}{2},n_1+n_2-2} = \pm t_{1-\frac{\alpha}{2},n_1+n_2-2} \tag{6.11}$$

where $t_{\frac{\alpha}{2},n_1+n_2-2}$ is the critical value, $t_{1-\frac{\alpha}{2},n_1+n_2-2}$ is the t-value that corresponds to the upper (or lower) tail area of the t-distribution with $n_1 + n_2 - 2$ degrees of freedom and a significance level of $\frac{\alpha}{2}$, and $\pm$ indicates whether the critical value is positive or negative, depending on whether it is a one-tailed or two-tailed test.

For a two-sample t-test with unequal variances, the critical value is found using the t-distribution with degrees of freedom approximated by the Welch-Satterthwaite equation:[38]

$$\nu \approx \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(\frac{s_1^2}{n_1})^2}{n_1-1} + \frac{(\frac{s_2^2}{n_2})^2}{n_2-1}} \tag{6.12}$$

where ν is the degrees of freedom, s_1^2 and s_2^2 are the sample variances, and n_1 and n_2 are the sample sizes. The formula for finding the critical value is:

$$t_{\frac{\alpha}{2},\nu} = \pm t_{1-\frac{\alpha}{2},\nu} \tag{6.13}$$

where $t_{\frac{\alpha}{2},\nu}$ is the critical value, $t_{1-\frac{\alpha}{2},\nu}$ is the t-value that corresponds to the upper (or lower) tail area of the t-distribution with ν degrees of freedom and a significance level of $\frac{\alpha}{2}$, and $\pm$ indicates whether the critical value is positive or negative, depending on whether it is a one-tailed or two-tailed test.

Example 6.1.3. An example of a t-test would be comparing the mean test scores of two classes of students to determine if there is a significant difference in their performance. Suppose we have the following data:

$$\text{Class 1: } n_1 = 25,\ x_1 = 85,\ s_1 = 10$$
$$\text{Class 2: } n_2 = 30,\ x_2 = 78,\ s_2 = 12$$

We want to test if the mean test scores of the two classes are significantly different at a 5% level of significance.

The null and alternative hypotheses are:

$$H_0 : \mu_1 = \mu_2$$
$$H_a : \mu_1 \neq \mu_2$$

To test whether the mean test scores of the two classes are significantly different, we can use a two-sample t-test with equal variances, since we do not have any prior information to suggest that the variances of the two populations are different.

The null hypothesis is that the mean test scores of the two classes are equal, while the alternative hypothesis is that they are not equal.

The pooled sample variance can be calculated as:

$$s_p = \sqrt{\frac{(25-1)(10)^2 + (30-1)(12)^2}{25+30-2}} \approx 11.13$$

The sample means are $\bar{x}_1 = 85$ and $\bar{x}_2 = 78$, and the sample sizes are $n_1 = 25$ and $n_2 = 30$.

We then calculate the t-score using the formula:

$$t = \frac{x_1 - x_2}{s_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

Substituting the values, we get:

$$t = \frac{85-78}{11.13\sqrt{\frac{1}{25} + \frac{1}{30}}} \approx 2.33$$

The degrees of freedom for the two-sample t-test with equal variances is:

$$df = n_1 + n_2 - 2 = 30 + 25 - 2 = 53$$

To test the null hypothesis at a 5% level of significance, we compare the test statistic to the critical value of the t-distribution with 53 degrees of freedom and a significance level of 0.05. Using the relevant formula or a precompiled t-table, we find the critical value to be approximately 2.009.

Since the test statistic (2.33) is greater than the critical value (2.009), we reject the null hypothesis and conclude that there is a significant difference in the mean test scores of the two classes at a 5% level of significance.

6.1.3 Confidence Intervals for Means

A ***confidence interval for the mean*** is a range of values that is likely to contain the true population mean with a certain degree of confidence. In other words, it is a statistical estimate of the range of values in which the population mean is likely to lie. Strictly speaking, confidence intervals are not the same as hypothesis tests, although they can be used to support or confirm the results of a hypothesis test.

To calculate a confidence interval for the mean, we need to know the sample mean, sample size, standard deviation, and the desired level of confidence. The level of confidence is typically chosen to be 90%, 95%, or 99%. For example, if we want to construct a 95% confidence interval for the mean, we would use a z-score of 1.96, which corresponds to the middle 95% of the standard normal distribution.

The formula for calculating a confidence interval for the mean is given as:

$$CI = \overline{X} \pm Z_{\frac{\alpha}{2}}\left(\frac{S}{\sqrt{n}}\right) \tag{6.14}$$

where $\overline{X}$ is the sample mean, S is the sample standard deviation, n is the sample size, Z is the critical value of the standard normal distribution for the desired level of confidence, and CI is the confidence interval.

The confidence interval for the mean is closely related to the t-test and the z-test, as they all involve the mean and the standard deviation of a sample, and are used to make inferences about the population parameters.

As we noted, z-test is used when the population standard deviation is known, or when the sample size is large enough to assume normality, while the t-test is used when the population standard deviation is unknown and must be estimated from the sample. Both tests use the standard error of the mean to calculate the test statistic, which is then compared to the appropriate critical value to determine the p-value and whether the null hypothesis should be rejected (see the relevant section on p-values in what follows).

The confidence interval for the mean can be used as an alternative or complementary approach to the t-test and z-test, as it provides an interval estimate of the population mean instead of a single point estimate, and can be more informative in some cases. The confidence interval can be used to determine whether the null hypothesis (such as a population mean being equal to a certain value) falls within the interval, and if not, it can be rejected.

Example 6.1.4. Suppose we want to compare the mean weight of two different groups of people: Group A and Group B.

Instead of conducting a t-test or z-test, we can calculate the confidence intervals for the mean weight of each group and see if they overlap. If the confidence intervals overlap, then we cannot conclude that there is a significant difference between the two

groups. If the confidence intervals do not overlap, then we can conclude that there is a significant difference between the two groups.

For example, let's say we have a sample of 50 individuals from Group A with a mean weight of 150 pounds and a standard deviation of 10 pounds, along with a sample of 50 individuals from Group B with a mean weight of 160 pounds and a standard deviation of 12 pounds. We want to determine if there is a significant difference in mean weight between the two groups using a 95% confidence interval.

We can calculate the 95% confidence interval for the mean weight of Group A as:

$$CI_A = 150 \pm 1.96 \times \left(\frac{10}{\sqrt{50}}\right)$$

$$CI_A = (146.19, 153.81)$$

And the 95% confidence interval for the mean weight of Group B as:

$$CI_B = 160 \pm 1.96 \times \left(\frac{12}{\sqrt{50}}\right)$$

$$CI_B = (155.68, 164.32)$$

Since the two confidence intervals do not overlap, we can conclude that there is a significant difference in mean weight between Group A and Group B.

6.1.4 Chi-Squared Tests

Chi-Squared Test is a statistical test that is used to determine whether there is a significant association between two categorical variables. The test is based on comparing the observed frequencies of data with the expected frequencies, assuming that the two variables are independent. The test provides a way to evaluate the null hypothesis that there is no association between the variables.

The Chi-Squared Test is different from z-tests and t-tests in that it is used to test for associations between *categorical* variables, while z-tests and t-tests are used to test for differences in means between continuous variables. In addition, the Chi-Squared Test is a non-parametric test, meaning that it does not assume a specific distribution for the data, while z-tests and t-tests are parametric tests and assume that the data follows a normal distribution.

There are several variants of the Chi-Squared test, including the Pearson Chi-Squared test, the Likelihood Ratio Chi-Squared test, and Fisher's Exact test. Each variant is used in different scenarios and has its own assumptions and limitations. Overall, the choice of which variant of the Chi-Squared test to use depends on the specific scenario and the assumptions and limitations of each test. If the sample size is large and the expected frequencies are not too small, the Pearson Chi-Squared test is the most appropriate test. If the sample size is small and the expected frequencies are small, either the Likelihood Ratio Chi-Squared test or Fisher's Exact test may be more appropriate. It is important to consider the specific scenario and the assumptions of each test when choosing which test to use.

Here we only mention the Pearson Chi-Squared Test. The ***Pearson Chi-Squared Test*** is used to determine whether there is a significant association between two or more categorical variables.

The formula for the Pearson Chi-Squared Test is as follows:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{6.15}$$

where:

- O_i is the observed frequency in the ith category
- E_i is the expected frequency in the ith category
- n is the number of categories

The test statistic χ^2 measures the difference between the observed and expected frequencies, standardized by the expected frequencies. If the observed frequencies are close to the expected frequencies, the test statistic will be small, indicating a good fit. If the observed frequencies are significantly different from the expected frequencies, the test statistic will be large, indicating a poor fit.

To perform the Pearson Chi-Squared Test, you first calculate the expected frequencies under the null hypothesis of no difference between the observed and expected frequencies. Then, you calculate the test statistic using the formula above and compare it to the critical value from the Chi-Squared distribution with $(n-1)$ degrees of freedom at the desired level of significance. If the test statistic is greater than the critical value, you reject the null hypothesis and conclude that there is a significant difference between the observed and expected frequencies.

The null hypothesis for the Chi-Squared Test is that there is no association between the two variables, and the alternative hypothesis is that there is a significant association between the variables. The significance level, denoted by α, is the probability of rejecting the null hypothesis when it is true.

Example 6.1.5. Suppose you work for a company that sells three different types of products: Product A, Product B, and Product C. The company recently conducted a marketing campaign to see if the pro-

portion of sales for each product would change. Before the campaign, the company sold 50% Product A, 30% Product B, and 20% Product C. After the campaign, the company sold 40% Product A, 35% Product B, and 25% Product C. The company wants to know if the campaign had a statistically significant effect on the proportions of sales for each product.

To test this, we can use the Pearson Chi-Squared Test. The null hypothesis is that the proportions of sales for each product are the same before and after the campaign, while the alternative hypothesis is that they are different. We can set the significance level to 5%.

To calculate the test statistic, we first need to calculate the expected frequencies under the null hypothesis. We can do this by multiplying the total number of sales before and after the campaign by the proportions of each product before and after the campaign, respectively. Suppose this gives us the following table:

	Before Campaign	After Campaign
Product A	150	120
Product B	90	105
Product C	60	75

To calculate the test statistic, we use the formula:

$$\chi^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

Plugging in the values from the table, we get:

$$\chi^2 = \frac{(75-60)^2}{60} + \frac{(105-90)^2}{90} + \frac{(120-150)^2}{150} = 10.67$$

We can then use a chi-squared distribution table, a calculator or just the CNF Chi-Square distribution function to find the p-value associated with this test statistic. With 2 degrees of freedom (since there are 3 categories and we used 1 degree of freedom to calculate the expected frequencies), the p-value is approximately 0.005, which is less than 0.05.[39] Therefore, we reject the null hypothesis and conclude that the marketing campaign had a statistically significant effect on the proportions of sales for each product.

6.1.5 F-Tests

The ***F-test*** is a statistical test used to compare the variances of two populations. It is based on the ratio of the variances of two samples and follows an F-distribution.

There are two main types of F-tests: one-sample F-tests and two-sample F-tests.

A ***one-sample F-test*** is used to determine whether the variance of a single population is equal to a hypothesized value. (The null hypothesis for a one-sample F-test is that the population variance is equal to a specific value, while the alternative hypothesis is that it is not equal to that value.)

The test statistic for a one-sample F-test is calculated by:

$$F = \frac{s^2}{\sigma_0^2} \tag{6.16}$$

where s^2 is the sample variance and σ_0^2 is the hypothesized population variance. The test statistic follows an F-distribution with $n-1$ degrees of freedom, where n is the sample size.

A ***two-sample F-test*** is used to compare the variances of two populations. (The null hypothesis for a two-sample F-test is that the variances of the two populations are equal, while the alternative hypothesis is that they are not equal.)

The test statistic for a two-sample F-test is calculated as:

$$F = \frac{s_1^2}{s_2^2} \tag{6.17}$$

where s_1^2 and s_2^2 are the sample variances of the two populations. The test statistic follows an F-distribution with $n_1 - 1$ degrees of freedom in the numerator and $n_2 - 1$ degrees of freedom in the denominator, where n_1 and n_2 are the sample sizes of the two populations.

In general, a higher value of the F-statistic indicates a greater difference in the variances of the populations being compared. The critical value of the F-statistic depends on the chosen level of significance and the degrees of freedom associated with the numerator and denominator of the F-distribution.

Example 6.1.6. Suppose a quality control manager wants to test whether the variance of the weights of a particular brand of cereal is equal to 0.02 ounces squared. The manager takes a random sample of 25 cereal boxes and records their weights. The sample has a variance of 0.0185 ounces squared.

To conduct a one-sample F-test, the manager would use the formula above to get:

$$F = \frac{0.0185}{0.02} = 0.925$$

Next, the manager would compare the calculated F-statistic (0.925) to the critical F-value for the test. Let's assume that the critical F-value for a significance level of 0.05 and 24 degrees of freedom (corresponding to a sample size of 25) is 2.064.

Since 0.925 is less than 2.064, we would fail to reject the null hypothesis that the population variance is equal to 0.02 ounces squared. In other words, there is not enough evidence to suggest that the variance of the weights of the cereal boxes is significantly different from the hypothesized value of 0.02 ounces squared.

Example 6.1.7. Suppose we have two samples, each containing $n_1 = 20$ and $n_2 = 25$ observations, respectively. Let's assume that the variances of the two populations are not equal, and we want to test if the means of the two populations are equal using a significance level of $\alpha = 0.05$. Here are the summary statistics for the two samples:

$$\text{Sample 1: } n_1 = 20, \bar{x}_1 = 10, s_1 = 3$$
$$\text{Sample 2: } n_2 = 25, \bar{x}_2 = 12, s_2 = 4$$

The null and alternative hypotheses for the test are as follows:

$$H_0 : \mu_1 = \mu_2$$
$$H_1 : \mu_1 \neq \mu_2$$

To conduct the F-test, we first calculate the sample variances and the test statistic as follows:

Sample variance for Sample 1: $s_1^2 = 3^2 = 9$

Sample variance for Sample 2: $s_2^2 = 4^2 = 16$

The test statistic for the F-test is calculated as:

$$F = \frac{s_1^2}{s_2^2} = \frac{9}{16} = 0.5625$$

The degrees of freedom for the numerator and denominator are $df_1 = n_1 - 1 = 19$ and $df_2 = n_2 - 1 = 24$, respectively.

To find the critical value from the F-distribution table, we need to use the significance level $\alpha = 0.05$ and the degrees of freedom $df_1 = 19$ and $df_2 = 24$. From the table, the critical value is $F_{0.025,19,24} = 2.39$.[40][41]

Since our calculated F-value of 0.5625 is less than the critical value of 2.39, we fail to reject the null hypothesis. We conclude that there is not enough evidence to suggest that the means of the two populations are different at the 5% significance level.

6.1.6 P-Values

In statistical hypothesis testing, the ***p-value*** is the probability of observing a test statistic as extreme as, or more extreme than, the one computed from the sample data, assuming that the null hypothesis is true.

The p-value is used to determine the statistical significance of the test results. If the p-value is less than the level of significance (usually 0.05), then we reject the null hypothesis and accept the alternative hypothesis. If the p-value is greater than the level of significance, then we fail to reject the null hypothesis.

To determine the p-value, it is crucial to know the distribution of the test statistic under the assumption that the null hypothesis is true. The cumulative distribution function (CDF) of the test statistic's distribution provides a tool to determine the p-value for a given sample.

1. For left-tailed tests, the p-value is simply the CDF of the test statistic's value x:
$$\text{p-value} = \text{cdf}(x) \tag{6.18}$$
2. For right-tailed tests, the p-value is:
$$\text{p-value} = 1 - \text{cdf}(x) \tag{6.19}$$
3. For two-tailed tests, the p-value is:
$$\text{p-value} = 2 \times \min\{\text{cdf}(x), 1 - \text{cdf}(x)\} \tag{6.20}$$

When the distribution of the test statistic under the null hypothesis is symmetric about 0, the calculation of the two-sided p-value can be simplified by taking the absolute value of x. In this case, the two-sided p-value is equal to:[42]

$$2 \times \text{cdf}(-|x|) = 2 - 2 \times \text{cdf}(|x|) \tag{6.21}$$

The p-value is a measure of the strength of the evidence against the null hypothesis. It represents the probability of obtaining a test statistic at least as extreme as the one computed from the sample data, assuming that the null hypothesis is true.

For example, imagine we studied the effect of a new drug and got a p-value of 0.04. This means that in 4% of similar studies, random chance alone would still be able to produce the value of the test statistic that we obtained, or a value even more extreme, even if the drug had no effect at all!

A low p-value (typically less than $\alpha = 0.05$) indicates that the observed effect is unlikely to be due to chance alone, and that we have strong evidence to reject the null hypothesis. This suggests that the alternative hypothesis is more likely to be true, and that the effect is real and meaningful.

On the other hand, a high p-value (typically greater than $\alpha = 0.05$) indicates that the observed effect is likely to be due to chance, and that we do not have strong evidence to reject the null hypothesis. This suggests that the effect may not be real or meaningful, and that further research may be needed to confirm or refute the hypothesis.

However, it is important to note that the interpretation of the p-value should always be accompanied by a careful consideration of the study design, sample size, and practical implications of the results. A statistically significant result (i.e., a low p-value) does not necessarily imply a clinically or practically important effect. Similarly, a non-significant result (i.e., a high p-value) does not necessarily imply the absence of an effect.

For example, suppose a pharmaceutical company is testing a new drug for a specific condition, and the study shows that the drug has a statistically significant effect (low p-value) in reducing symptoms of the condition. However, upon further analysis, the effect size is found to be small, and the cost of the drug is high. In this case, the clinical importance of the result may be called into question, as the practical benefits of the drug may not outweigh its costs.

Note that even though a significance level of 0.05 is the most common value to compare the p-value to, there's nothing magical about it. In fact, the choice of $\alpha = 0.05$ as a threshold for statistical significance is somewhat arbitrary and has been historically used as a standard convention in many fields. However, this threshold is not universally applicable, and it should not be considered as a fixed rule.

The appropriate choice of α depends on several factors, including the field of study, the nature of the research question, the study design, and the consequences of Type I and Type II errors. In general, a lower α level (e.g., 0.01) may be more appropriate when the consequences of a false positive result are severe, such as in medical or safety-critical contexts. On the other hand, a higher α level (e.g., 0.10) may be more appropriate when the consequences of a false negative result are more severe, such as in exploratory research or when the cost of data collection is high. It is important to select an appropriate alpha level in advance and to report it transparently in the study design to avoid post-hoc adjustments that may lead to biased results.

Example 6.1.8. Suppose we have a sample of 10 observations with a sample mean of 75 and a sample standard deviation of 5. We want to test the null hypothesis that the population mean is equal to 70 using a two-tailed t-test with a significance level of 0.05.

Using the formula for the t-statistic, we calculate:

$$t = \frac{75 - 70}{\frac{5}{\sqrt{10}}} = 2.236$$

The degrees of freedom for the test is $n - 1 = 9$. Looking up the p-value for a two-tailed t-test with 9 degrees of freedom and a t-statistic of 2.236 in a t-table, we find the p-value to be approximately 0.047.

Since the p-value is less than the significance level of 0.05, we reject the null hypothesis and conclude that there is sufficient evidence to suggest that the population mean is not equal to 70.

Example 6.1.9. Suppose we have two independent samples, each with a sample size of 10. The first sample has a sample variance of 5 and the second sample has a sample variance of 8. We want to test the null hypothesis that the population variances are equal using a two-tailed F-test with a significance level of 0.05.

Using the formula for the F-statistic, we have:

$$F = \frac{5}{8} = 0.625$$

The degrees of freedom for the test are $n_1 - 1 = 9$ and $n_2 - 1 = 9$. Looking up the p-value for a two-tailed F-test with 9 and 9 degrees of freedom and an F-statistic of 0.625 in an F-table, we find the p-value to be approximately 0.821.

Since the p-value is greater than the significance level of 0.05, we fail to reject the null hypothesis and conclude that there is not sufficient evidence to suggest that the population variances are different.

6.1.7 A|B Testing

A|B testing is a statistical technique used to compare the effectiveness of two different versions of a product, webpage, or marketing campaign. It involves randomly assigning participants into two groups: Group A, which experiences the control version, and Group B, which experiences the experimental version. The results from both groups are then compared to determine which version performs better.

To perform A|B testing, we can use a hypothesis testing framework. The null hypothesis states that there is no significant difference between the two versions, while the alternative hypothesis states that there is a significant difference. We can then calculate a p-value using a suitable statistical test (such as a z-test, t-test, or chi-squared test) and compare it to our chosen significance level (usually 0.05). If the p-value is less than the significance level, we reject the null hypothesis and conclude that there is a significant difference between the two versions.

Example 6.1.10. For example, let's say a company wants to compare the effectiveness of two different webpage designs in terms of click-through rates. They randomly assign 500 participants to Group A,

which experiences the original design, and 500 participants to Group B, which experiences the new design. After collecting data on the number of clicks from each group, they calculate the click-through rate for each group and perform a hypothesis test using a z-test. They find that the p-value is 0.03, which is less than the significance level of 0.05, so they reject the null hypothesis and conclude that the new design has a significantly different click-through rate than the original design.

Notes

[29]The image above is taken from this paper: https://arxiv.org/pdf/1905.00734.pdf

[30]The image is taken from https://vwo.com/blog/errors-in-ab-testing/ and enhanced to meet our needs.

[31]The following diagram is taken from https://en.wikipedia.org/wiki/Standard_score

[32]The following three images are taken from this online article due to their nice graphics: https://www.cuemath.com/data/hypothesis-testing/

[33]You could use the following code snippet in Python to get the number:

```
from scipy.stats import norm
1 - norm.cdf(0.089)
```

[34]You could use the following code snippet in Python to get the number:

```
from scipy.stats import norm
1 - norm.cdf(1.79)
```

[35]You could use the following code snippet in Python to get the number:

```
from scipy.stats import norm
norm.ppf(1 - 0.05/2).
```

Note that 'norm.ppf' and 'norm.cdf' are both methods in the 'scipy.stats.norm' module that are used to work with the normal distribution.

norm.ppf stands for percent point function, also known as the inverse cumulative distribution function. It takes a probability level as an argument and returns the z-score (or x-value) at which that probability level is reached in a standard normal distribution. For example, if we want to find the z-score that corresponds to the 95th percentile in a standard normal distribution, we can use norm.ppf(0.95). We use norm.ppf(1 - alpha/2) to determine the critical value for a two-tailed test because we want to find the z-score that corresponds to the probability level that includes both tails of the distribution.

norm.cdf, on the other hand, stands for cumulative distribution function. It takes a z-score (or x-value) as an argument and returns the probability of observing a value less than or equal to that z-score (or x-value) in a standard normal distribution. For example, if we want to find the probability of observing a z-score less than or equal to 1.96 in a standard normal distribution, we can use norm.cdf(1.96).

[36]By 'precision', here we mean the degree to which the t-test can accurately distinguish between the two sample means. When the variances of the two populations are equal, the pooled standard deviation provides a more accurate estimate of the population standard deviation than using the individual sample standard deviations separately. This leads to a more accurate estimate of the standard error of the mean difference, which in turn leads to a more precise t-test statistic.

[37]The following image is taken from https://www.scribbr.com/statistics/degrees-of-freedom/. I recommend checking out the article for more analogies and examples to illustrate degrees of freedom.

[38]Check out this Wiki article on the Welch-Satterthwaite equation: https://en.wikipedia.org/wiki/Welch-Satterthwaite_equation

[39]You could also use the following Python code snippets to get a similar result. First `import scipy.stats as stats` and then run this code: `1 - stats.chi2.cdf(10.67, 2)`

[40]Check out the table here, for example: http://www.socr.ucla.edu/Applets.dir/F_Table.html

[41]Alternatively you could use the following Python code to get this result: first `import scipy.stats as stats`, then `stats.f.ppf(q=1-0.05, dfn=19, dfd=24)`.

[42]The equality $2 \times \text{cdf}(-|x|) = 2 - 2 \times \text{cdf}(|x|)$ follows from the fact that the two-sided p-value is equal to twice the minimum of the probabilities of observing a test statistic as extreme or more extreme than the one computed from the sample data, in either direction. Using the absolute value of the test statistic x ensures that we capture the probabilities in both tails of the distribution, without losing information about the direction of the deviation from the null hypothesis. Therefore, the two-sided p-value can be expressed as: p-value = $2 \times \min\{\text{cdf}(x), 1 - \text{cdf}(x)\} = 2 \times \min\{\text{cdf}(|x|), 1 - \text{cdf}(|x|)\}$. Since the distribution of the test statistic under the null hypothesis is symmetric about 0, we can simplify the expression above by noting that $\text{cdf}(-|x|) = 1 - \text{cdf}(|x|)$, since the probabilities in the left tail of the distribution are equal to the probabilities in the right tail of the distribution. Therefore, we can write: p-value = $2 \times \min\{\text{cdf}(|x|), 1 - \text{cdf}(|x|)\} = 2 \times \text{cdf}(-|x|) = 2 - 2 \times \text{cdf}(|x|)$, which is the desired equality.

6.2 Further Reading

Here are a few useful resources on statistics, particularly focused on inferential statistics and hypothesis testing:

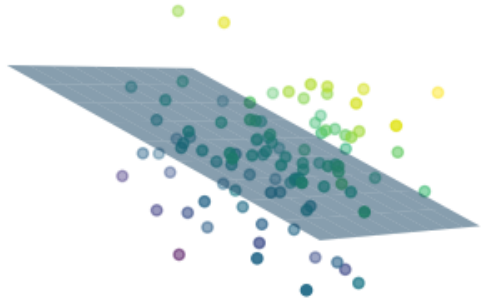
1. *Statistics for Business & Economics*, by Anderson, et. al [3]. This book covers a broad range of statistical topics, including descriptive statistics, probability theory, statistical inference, and regression analysis, with a focus on real-world applications. The examples and exercises are practical and relevant to business and economics, and the authors provide clear explanations for conducting statistical analysis.
2. *Statistical Inference,* by George Casella and Roger Berger [9]. This graduate-level textbook covers the theory of statistical inference, including estimation, hypothesis testing, and confidence intervals. It also includes coverage of Bayesian inference and decision theory.
3. *Hypothesis Testing: A Visual Introduction To Statistical Significance*, by Scott Hartshorn [15]. This introductory textbook covers the basics of hypothesis testing, including the use of p-values, confidence intervals, and effect sizes. The book emphasizes the importance of visualizations in understanding statistical concepts.

Part II

Mathematics in Machine Learning

Chapter 7

Linear Regression



7.1 Introduction

Linear regression is a statistical method used in data science to establish a relationship between one or more independent variables and a dependent variable. The dependent variable is the outcome or result we are interested in predicting or explaining, while the independent variables are the predictors or factors that can influence the outcome. Linear regression models the relationship between the independent variables and dependent variables as a linear equation (a straight line, a plane or a multidimensional surface), making it a popular tool for predicting and explaining numerical outcomes.

The term 'regression' comes from the observation made by Sir Francis Galton, a 19th-century English mathematician and scientist, who noticed that the heights of children tended to regress towards the average height of their parents. He called this phenomenon "regression towards mediocrity," which later inspired the term 'regression' in statistics. In the context of linear regression, the term refers to the idea that the model tries to find the 'best fit" line that minimizes the distance between the predicted values and the actual values, which can be thought of as "regressing" towards the mean of the data.

There are two types of linear regression: simple linear regression and multiple linear regression. Simple linear regression models the relationship between one independent variable and one dependent variable, while multiple linear regression models the relationship between two or more independent variables and one dependent variable. In both cases, the objective is to find the best-fit line that represents the relationship between the variables.

7.2 Simple Linear Regression

Simple linear regression models the relationship between a *single* independent variable and a dependent variable. The goal is to find the line of best fit that represents the relationship between the variables. The equation for the line of best fit is:

$$\hat{y} = \theta_0 + \theta_1 x \tag{7.1}$$

where $\hat{y}$ is the predicted value of the dependent variable, x is the independent variable, θ_0 is the intercept. i.e., the point where the line of best fit intersects the y-axis, and θ_1 is the slope of the line.

7.2.1 Cost Function

In linear regression, the ***cost function*** is used to measure the difference between the predicted values of the model and the actual values of the target variable.

The most commonly used cost function in linear regression is the ***mean squared error*** (***MSE***) function, which calculates the average squared difference between the predicted values and the actual values.

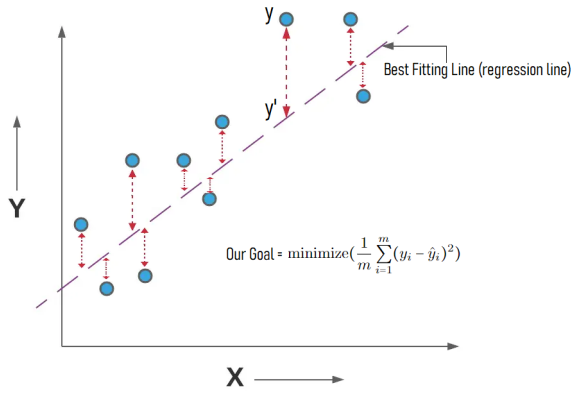
The formula for the MSE cost function in linear regression is:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \tag{7.2}$$

where:

- $J(\theta)$ is the cost function,
- m is the number of training examples,
- $h_\theta(x^{(i)}) = \hat{y} = \theta_0 + \theta_1 x$ is the predicted value for the ith training example,
- $y^{(i)}$ is the actual value for the ith training example.[43]

The goal of linear regression is to minimize the cost function by finding the values of θ_0 and θ_1 that produce the smallest MSE.



7.2.2 Gradient Descent

Gradient descent is an optimization algorithm commonly used in machine learning to minimize a cost function. The idea behind gradient descent is to iteratively adjust the parameters of a model in the direction of the steepest descent of the cost function until a minimum is reached.

Here's how the algorithm works:

1. Initialize the parameters of the model (e.g., weights and biases) to some random values.
2. Calculate the gradient of the cost function with respect to the parameters. This tells us how much the cost function changes for small changes in the parameters.
3. Update the parameters by subtracting a fraction of the gradient from the current values. This moves the parameters in the direction of steepest descent of the cost function. The fraction by which the gradient is multiplied is known as the learning rate, and it determines how quickly the algorithm converges to the minimum.
4. Repeat steps 2 and 3 until the cost function reaches a minimum or some stopping criterion is met.

By iteratively adjusting the parameters in the direction of the steepest descent of the cost function, gradient descent can find the optimal values of the parameters that minimize the cost function. There are different variations of gradient descent, such as batch gradient descent, stochastic gradient descent, and mini-batch gradient descent, which differ in how they compute the gradient and update the parameters.[44]

The formal notation for gradient descent can be expressed as follows:

Repeat until convergence

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \tag{7.3}$$

where the partial derivative $\frac{\partial}{\partial \theta_j} J(\theta)$ measures the sensitivity of the cost function $J(\theta)$ to changes in the jth parameter θ_j, and the ***learning rate*** α determines the step size of the update. The goal is to move the parameters in the direction of the steepest descent of the cost function, which is achieved by subtracting the gradient from the current parameter values.

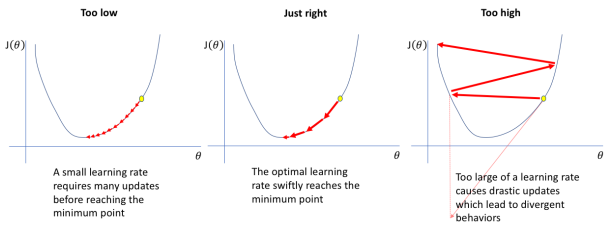
The partial derivatives of the cost function with respect to θ_0 and θ_1 are:

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)}) \tag{7.4}$$

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)} \tag{7.5}$$

By repeatedly updating the values of θ_0 and θ_1 using the gradient descent algorithm, the values of θ_0 and θ_1 that minimize $J(\theta)$ can be found. These values can then be used to make predictions for new values of the independent variable.

Before seeing an example, a remark regarding the learning rate, α, is in order. The learning rate determines the magnitude of the step taken in each iteration toward the optimal solution. If the learning rate is too high, the optimization algorithm may 'overshoot' the optimal solution and fail to converge. If the learning rate is too low, the optimization algorithm may take a long time to converge, especially for large datasets. So one might need to look for a balance in choosing α.[45]



Example 7.2.1. Suppose we have a dataset of 5 houses with their sizes (in square feet) and prices (in thousands of dollars) as follows:

Size (sq. ft.)	Price (1000s of $)
1050	65
1502	89
2000	120
2540	135
2800	150
5212	254
2123	134
4513	245
4029	210

We want to fit a linear regression model to this data to predict the price of a house based on its size. We assume that the relationship between the size and price can be modeled using a linear function of the form:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

where θ_0 is the intercept (the price of a house with size 0) and θ_1 is the slope (the increase in price per unit increase in size).

To find the optimal values of the model parameters θ_0 and θ_1, we need to minimize the cost function, which is defined as:

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

where m is the number of training examples, $x^{(i)}$ is the size of the ith house, $y^{(i)}$ is the price of the ith house, and $h_\theta(x^{(i)})$ is the predicted price of the ith house based on the current values of the model parameters.

We use gradient descent to $J(\theta)$. We start with some initial values for θ_0 and θ_1, and then iteratively update them using the above equations until convergence. The algorithm stops when the cost function no longer decreases or reaches a predefined threshold.

For example, let's start with $\theta_0 = 0$ and $\theta_1 = 0$ as the initial values, and $\alpha = 0.01$ as the learning rate. We can update the parameters using the gradient descent algorithm as follows:

1. Compute the predicted values for each training example:

$$h_\theta(x^{(i)}) = 0 + 0\theta_1 x^{(i)} = 0$$

2. Compute the error term for each example:

$$error^{(i)} = 0 - y^{(i)}$$

3. Update the parameter values:

$$\theta_0 := 0 - 0.01\frac{1}{9}\sum_{i=1}^{9}(-\hat{y})^{(i)}$$

$$\theta_1 := 0 - 0.01\frac{1}{9}\sum_{i=1}^{9}(-\hat{y})^{(i)}x^{(i)}$$

4. Compute the cost function using the updated parameters:

$$J(\theta) = \frac{1}{2\times 9}\sum_{i=1}^{9}(h_\theta(x^{(i)}) - y^{(i)})^2$$

5. Repeat steps 1-4 until convergence (i.e., until the cost function no longer decreases or reaches a predefined threshold).

Using the above algorithm, we can update the parameters multiple times until we reach convergence. After many iterations, the algorithm converges to the following parameter values:

$$\theta_0 = 22.60,\ \theta_1 = 0.046$$

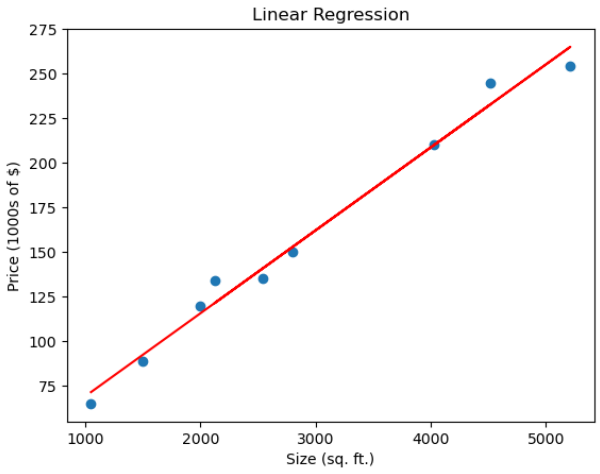
These values represent the best-fitting line for the given data, which can be written as:[46]

$$h_\theta(x) = 22.60 + 0.046$$

Finally, we can compute the cost function using the optimized parameter values to evaluate the performance of the model. The cost function for the optimized model (MSE) is:

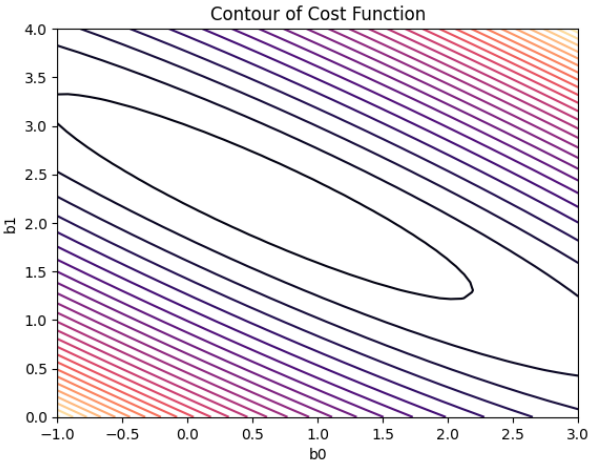
$$J(\theta) = 61.24$$

This means that the model has a mean squared error of 0.33, which is relatively small and indicates a good fit for the data.



We can visualize the evolution of the θ_0 and θ_1 simultaneously for a cost function using what's known as a contour plot. In general, a contour plot is a graph that displays the relationship between two independent variables and a dependent variable in a regression model. It shows the values of the dependent variable for combinations of the independent variables.

In the context of linear regression, the contour plot provides a visual representation of the cost function surface and helps to identify the optimal combination of b0 and b1 that minimize the cost. The steepest descent direction for gradient descent is always perpendicular to the contour lines, which means that gradient descent will converge to the minimum of the cost function by following the direction of the steepest descent.



Contour plots can be used in conjunction with regression analysis to statistically test the relationships between variables. The contour lines represent different values of the cost function, with the innermost lines representing lower costs and the outermost lines representing higher costs. The minimum of the cost function corresponds to the optimal values of θ_0 and θ_1 that minimize the difference between the predicted values and the actual values of y.

7.3 Normal Equation

We just learned about gradient descent as an algorithm that indirectly, through many iterations, finds the optimal parameters for our model. One might wonder if this could've been done directly. It can, though what is called the 'normal equation'.

The ***normal equation*** for linear regression is a *direct* solution for finding the optimal coefficients for a linear regression model. It involves finding the value of the coefficient vector θ that minimizes the sum of squared errors between the predicted values $\hat{\mathbf{y}}$ and the actual values $\mathbf{y}$ in the training data.

Mathematically, the normal equation can be written as:

$$\theta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{7.6}$$

where $\mathbf{X}$ is the matrix of independent variables with each row representing a training example, $\mathbf{y}$ is the vector of dependent variables, and θ is the vector of coefficients.

The normal equation provides a computationally efficient way to find the optimal coefficients without having to use an iterative optimization algorithm like gradient descent. However, it's important to note that the normal equation may not always be the best approach, especially for very large datasets. In such cases, gradient descent may be more suitable as it can be faster and more scalable. Additionally, the normal equation may not work if the matrix $\mathbf{X}^T\mathbf{X}$ is singular or not invertible, in which case other techniques like ridge regression or lasso regression may be used.

We close this section by showing how can the normal equation be obtained by solving for the optimal coefficients in linear regression. (You don't need to read and understand this, but doing so will help you brush up your linear algebra and calculus skills, and see how a mathematical proof is done!)

Proof.

The normal equation for linear regression is derived by setting the derivative of the cost function with respect to the parameters to zero. That is:

$$\nabla_\theta J(\theta) = 0 \tag{7.7}$$

where $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ and $\nabla_\theta J(\theta)$ is the gradient of the cost function with respect to the parameters. We can expand this equation as:

$$\frac{\partial}{\partial\theta_j}\left(\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2\right) = 0$$

for all $j = 0, 1, ..., n$, where n is the number of features. This equation represents the condition for finding the optimal parameters θ of a linear regression model, where $h_\theta(x^{(i)})$ is the predicted value for the i-th training example $x^{(i)}$, and $y^{(i)}$ is the corresponding true label.

Taking the derivative with respect to θ_j, we obtain:

$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} = 0$$

Substituting $h_\theta(x^{(i)}) = \theta_0+\theta_1 x_1^{(i)}+\theta_2 x_2^{(i)}+\cdots+\theta_n x_n^{(i)}$, we get:

$$\frac{1}{m}\sum_{i=1}^{m}(\theta_0+\theta_1 x_1^{(i)}+\theta_2 x_2^{(i)}+\cdots+\theta_n x_n^{(i)}-y^{(i)})x_j^{(i)}=0$$

This equation can be written in matrix form as:

$$\frac{1}{m}X^T(h_\theta(x)-y)=0 \tag{7.8}$$

where $\mathbf{X}$ is the $m\times(n+1)$ design matrix, with each row representing a training example and each column representing a feature, $\mathbf{y}$ is the $m\times 1$ vector of true labels, and $h_\theta(x)$ is the $m\times 1$ vector of predicted values.

To be more specific, $\mathbf{X}$ is

$$X=\begin{bmatrix}1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)}\\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & x_1^{(m)} & x_2^{(m)} & \dots & x_n^{(m)}\end{bmatrix}$$

$h_\theta(x)$ is an $m\times 1$ column vector given by:

$$h_\theta(x)=\begin{bmatrix}h_\theta(x^{(1)})\\ h_\theta(x^{(2)})\\ \vdots\\ h_\theta(x^{(m)})\end{bmatrix}=\begin{bmatrix}\theta_0+\theta_1 x_1^{(1)}+\cdots+\theta_n x_n^{(1)}\\ \theta_0+\theta_1 x_1^{(2)}+\cdots+\theta_n x_n^{(2)}\\ \vdots\\ \theta_0+\theta_1 x_1^{(m)}+\cdots+\theta_n x_n^{(m)}\end{bmatrix}$$

and $\mathbf{y}$ is an $m\times 1$ column vector of true labels:

$$\mathbf{y}=\begin{bmatrix}y^{(1)}\\ y^{(2)}\\ \vdots\\ y^{(m)}\end{bmatrix}$$

The transpose of $\mathbf{X}$ is an $(n+1)\times m$ matrix:

$$\mathbf{X}^T=\begin{bmatrix}1 & 1 & \dots & 1\\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)}\\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)}\\ \vdots & \vdots & \ddots & \vdots\\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)}\end{bmatrix}$$

Therefore, $\frac{1}{m}\mathbf{X}^T(h_\theta(x)-y)$ is an $(n+1)\times 1$ column vector given by:

$$\frac{1}{m}\mathbf{X}^T(h_\theta(x)-y)=\begin{bmatrix}\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})\\ \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_1^{(i)}\\ \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_2^{(i)}\\ \vdots\\ \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_n^{(i)}\end{bmatrix}$$

where each element corresponds to the derivative of the cost function with respect to the corresponding parameter θ_j.

Multiplying both sides of the equation (7.8) by X^T, we get:

$$\frac{1}{m}\mathbf{X}^T\mathbf{X}\theta-\frac{1}{m}\mathbf{X}^T\mathbf{y}=0$$

Adding $\frac{1}{m}\mathbf{X}^T y$ to both sides, we obtain:

$$\frac{1}{m}\mathbf{X}^T\mathbf{X}\theta=\frac{1}{m}\mathbf{X}^T\mathbf{y}$$

Multiplying both sides by m, we get:

$$\mathbf{X}^T\mathbf{X}\theta=\mathbf{X}^T\mathbf{y}$$

which is the system of $n+1$ equations in matrix form that we were looking for.

The solution to this equation is:

$$\theta=(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

This is known as the normal equation and provides a closed-form solution for the optimal values of the parameters that minimize the cost function. □

7.4 Multiple Linear Regression

Multiple linear regression is a generalization of simple linear regression, where potentially more than one predictor variable is at stake. It assumes that the relationship between the response variable and each predictor variable is linear.

Things are quite similar to simple linear regression here; it's just that we're dealing with many more parameters, and the best fit may not be a line, but a plane or a higher-dimensional surface.

More specifically, the multiple linear regression model can be expressed as:

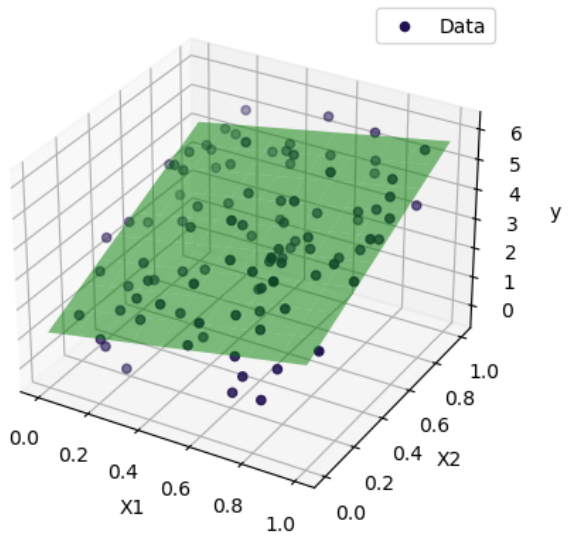
$$\hat{y}=\theta_0+\theta_1 x_1+\theta_2 x_2+...+\theta_p x_p \tag{7.9}$$

where $\hat{y}$ is the response variable, x_1, x_2, ..., x_p are the predictor variables, and θ_0, θ_1, θ_2, ..., θ_p are the coefficients or parameters to be estimated.

The cost function for multiple linear regression is the same for simple linear regression, given by:

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2$$

The only difference is that we have potentially more variables that define our cost function now.



The goal of multiple linear regression is to find the values of the coefficients θ_0, θ_1, θ_2, ..., θ_p that minimize the cost function $J(\theta)$. This can be done using gradient descent, which iteratively updates the values of the coefficients as:

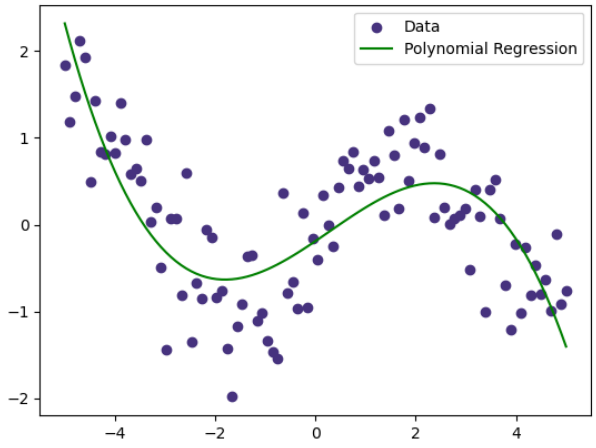
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

where α is the learning rate, and j is the index of the coefficient being updated. Again, this is essentially the same for simple regression, except that we might have more than just θ_0 and θ_1 to work with.

7.5 Polynomial Regression

Polynomial linear regression is a type of regression analysis where the relationship between the independent variable x and dependent variable y is modeled as an nth degree polynomial. It is an extension of multiple linear regression, where the variables take powers of potentially more than 1.

In some cases, we may also use polynomial regression to model interaction effects between two or more independent variables. In particular, if the relationship between x and y is not a straight line but can be approximated by a curve, we use polynomial linear regression. For example, in the case of stock prices, the relationship between the time (independent variable) and the stock price (dependent variable) may not be linear but can be approximated by a polynomial function.



The polynomial linear regression equation is given by:

$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + ... + \theta_n x^n \qquad (7.10)$$

where y is the dependent variable, x is the independent variable, θ_0 is the intercept, θ_1 to θ_n are the coefficients and n is the degree of the polynomial.

The cost function and gradient descent for polynomial linear regression are the same as those for multiple regression, with the predicting variables potentially having different powers than just 1.

7.6 Python Implementations

We now display some code samples in Python for implementing linear and polynomial regression.

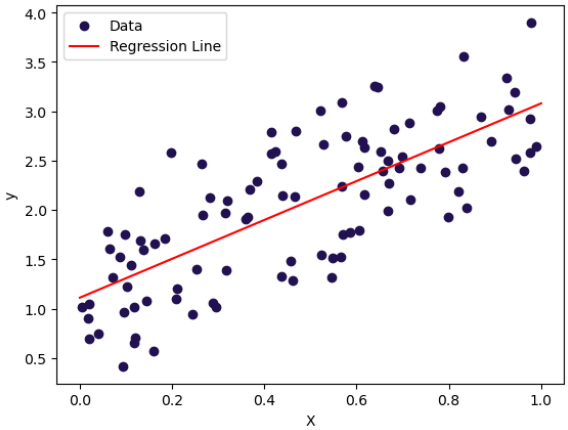
7.6.1 Simple Linear Regression

In this example, we generate some random data with a single independent variable (`X`) and one dependent variable (`y`). We train a linear regression model on the data using scikit-learn's LinearRegression class, which automatically computes the slope and intercept of the regression line using the least squares method. We then use the trained model to compute the predicted values for the regression line. Finally, we plot the original data and the regression line. You can run these codes in any Python environment such as Jupyter Notebook or Google Colab:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Generate some random data
np.random.seed(0)
n_samples = 100
X = np.random.rand(n_samples)
y = 1 + 2*X + np.random.normal(0, 0.5, n_samples)

# Train a linear regression model on the data
model = LinearRegression()
model.fit(X.reshape(-1, 1), y)

# Compute the slope and intercept of the regression line
slope = model.coef_[0]
intercept = model.intercept_

# Compute the loss function
y_pred = model.predict(X.reshape(-1, 1))
mse = mean_squared_error(y, y_pred)

# Compute the predicted values for the regression line
X_line = np.linspace(0, 1, 10)
y_line = model.predict(X_line.reshape(-1, 1))

# Plot the data and the regression line
plt.scatter(X, y, label='Data')
plt.plot(X_line, y_line, label='Regression Line', color='red')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

print(f"Slope: {slope}")
print(f"Intercept: {intercept}")
print(f"Mean Squared Error: {mse}")
```

Here are the import plot and prints:



```
Slope: 2.002914678851789
Intercept: 0.9698728752768302
Mean Squared Error: 0.2366107863732744
```

Well, this was fun! Note, that scikit-learn's `LinearRegression` does not use gradient descent to calculate the cost function and its optimized version. Instead, it uses the *Ordinary Least Squares* (OLS) method to estimate the coefficients of the linear regression model. The OLS method involves minimizing the sum of squared errors between the predicted values and the actual values of the target variable.

In contrast to gradient descent, the OLS method has a closed-form solution that allows the coefficients to be directly calculated without the need for iterative optimization. This makes it a more computationally efficient method for solving linear regression problems, especially when the number of features (i.e., independent variables) is small. Scikit-learn's `LinearRegression` implementation also includes some optimizations such as matrix operations that further improve its computational efficiency.

While this does most of what we need from linear regression, one might have a temptation to gain even more. For instance, as of now, we cannot access see what the iterations of the OLS look like, and how the parameters θ_0 and θ_1 evolve over iterations. This information just isn't shared with us.

But what if we wanted to access them? Well, we could code up the whole thing from scratch, with much more granular control over the parameters of the model. In particular, let's recreate the same code from scratch, without the use of Scikit-learn at any stage. Below we will do this using our beloved gradient descent:

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the number of samples and generate random data
np.random.seed(0)
n_samples = 100
X = np.random.rand(n_samples)
y = 1 + 2*X + np.random.normal(scale=0.5, size=n_samples)

# Define the coefficients function using gradient descent
def coefficients(X, y, learning_rate, n_iterations):
    n = len(X)
    b0, b1 = 0.0, 0.0
    cost_history = []
    b0_history = []
    b1_history = []
    for i in range(n_iterations):
        y_pred = b0 + b1 * X
        error = y_pred - y
        b0 -= learning_rate * (1/n) * np.sum(error)
        b1 -= learning_rate * (1/n) * np.sum(error * X)
        cost = np.sum((y - y_pred) ** 2) / float(n)
        cost_history.append(cost)
        b0_history.append(b0)
        b1_history.append(b1)
        print(f"Iteration {i+1}: b0={b0:.4f}, b1={b1:.4f}, cost={cost:.4f}")
    return b0, b1, cost_history, b0_history, b1_history

# Choose the learning rate and number of iterations
learning_rate = 0.01
n_iterations = 270
```

```
# Compute the coefficients of the regression line
intercept, slope, cost_history, b0_history, b1_history = coefficients(X, y, learning_rate,
n_iterations)
```

This outputs the following:

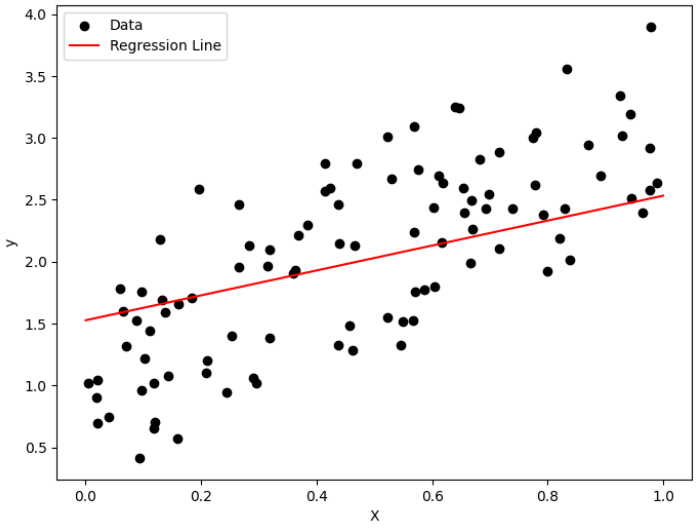
```
Iteration 1: b0=0.0204, b1=0.0113, cost=4.7389
Iteration 2: b0=0.0406, b1=0.0224, cost=4.6308
Iteration 3: b0=0.0605, b1=0.0335, cost=4.5252
Iteration 4: b0=0.0801, b1=0.0444, cost=4.4223
Iteration 5: b0=0.0995, b1=0.0552, cost=4.3219
Iteration 6: b0=0.1187, b1=0.0658, cost=4.2240
Iteration 7: b0=0.1376, b1=0.0763, cost=4.1284
Iteration 8: b0=0.1563, b1=0.0867, cost=4.0353
Iteration 9: b0=0.1747, b1=0.0970, cost=3.9444
Iteration 10: b0=0.1930, b1=0.1072, cost=3.8558
Iteration 11: b0=0.2109, b1=0.1172, cost=3.7693
Iteration 12: b0=0.2287, b1=0.1272, cost=3.6850
Iteration 13: b0=0.2462, b1=0.1370, cost=3.6027
Iteration 14: b0=0.2635, b1=0.1467, cost=3.5225
Iteration 15: b0=0.2806, b1=0.1563, cost=3.4442
Iteration 16: b0=0.2975, b1=0.1658, cost=3.3679
Iteration 17: b0=0.3142, b1=0.1751, cost=3.2934
Iteration 18: b0=0.3306, b1=0.1844, cost=3.2208
Iteration 19: b0=0.3468, b1=0.1936, cost=3.1499
Iteration 20: b0=0.3629, b1=0.2026, cost=3.0808
.
.
.
Iteration 185: b0=1.4447, b1=0.8961, cost=0.3750
Iteration 186: b0=1.4464, b1=0.8979, cost=0.3737
Iteration 187: b0=1.4481, b1=0.8995, cost=0.3726
Iteration 188: b0=1.4498, b1=0.9012, cost=0.3714
Iteration 189: b0=1.4515, b1=0.9029, cost=0.3703
Iteration 190: b0=1.4531, b1=0.9046, cost=0.3692
Iteration 191: b0=1.4547, b1=0.9062, cost=0.3681
Iteration 192: b0=1.4563, b1=0.9078, cost=0.3670
Iteration 193: b0=1.4579, b1=0.9095, cost=0.3660
Iteration 194: b0=1.4594, b1=0.9111, cost=0.3650
Iteration 195: b0=1.4609, b1=0.9127, cost=0.3640
Iteration 196: b0=1.4624, b1=0.9142, cost=0.3631
Iteration 197: b0=1.4639, b1=0.9158, cost=0.3621
Iteration 198: b0=1.4653, b1=0.9174, cost=0.3612
Iteration 199: b0=1.4668, b1=0.9189, cost=0.3603
Iteration 200: b0=1.4682, b1=0.9205, cost=0.3594
Iteration 201: b0=1.4696, b1=0.9220, cost=0.3586
Iteration 202: b0=1.4709, b1=0.9235, cost=0.3577
Iteration 203: b0=1.4723, b1=0.9250, cost=0.3569
Iteration 204: b0=1.4736, b1=0.9265, cost=0.3561
Iteration 205: b0=1.4749, b1=0.9280, cost=0.3553
Iteration 206: b0=1.4762, b1=0.9295, cost=0.3545
Iteration 207: b0=1.4774, b1=0.9309, cost=0.3538
Iteration 208: b0=1.4787, b1=0.9324, cost=0.3530
Iteration 209: b0=1.4799, b1=0.9338, cost=0.3523
Iteration 210: b0=1.4811, b1=0.9352, cost=0.3516
Iteration 211: b0=1.4823, b1=0.9366, cost=0.3509
Iteration 212: b0=1.4834, b1=0.9381, cost=0.3502
Iteration 213: b0=1.4846, b1=0.9395, cost=0.3495
Iteration 214: b0=1.4857, b1=0.9408, cost=0.3489
Iteration 215: b0=1.4868, b1=0.9422, cost=0.3482
Iteration 216: b0=1.4879, b1=0.9436, cost=0.3476
Iteration 217: b0=1.4890, b1=0.9450, cost=0.3470
Iteration 218: b0=1.4901, b1=0.9463, cost=0.3464
Iteration 219: b0=1.4911, b1=0.9477, cost=0.3458
Iteration 220: b0=1.4921, b1=0.9490, cost=0.3453
Iteration 221: b0=1.4931, b1=0.9503, cost=0.3447
Iteration 222: b0=1.4941, b1=0.9516, cost=0.3441
```

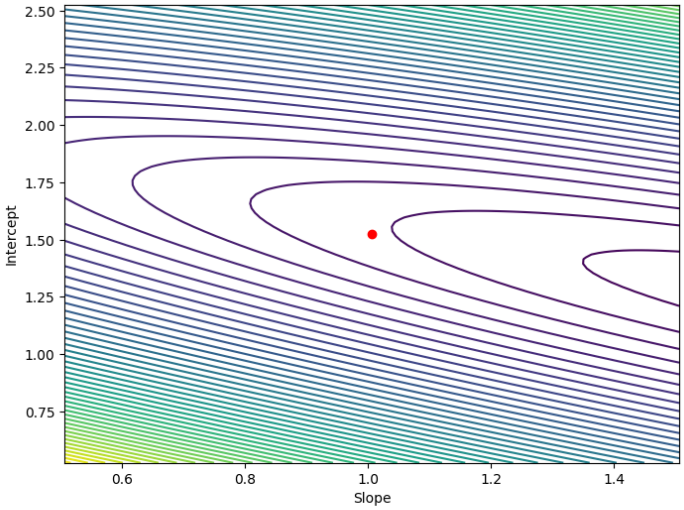
```
Iteration 223: b0=1.4951, b1=0.9529, cost=0.3436
Iteration 224: b0=1.4961, b1=0.9542, cost=0.3431
Iteration 225: b0=1.4970, b1=0.9555, cost=0.3425
Iteration 226: b0=1.4979, b1=0.9568, cost=0.3420
Iteration 227: b0=1.4989, b1=0.9581, cost=0.3415
Iteration 228: b0=1.4998, b1=0.9593, cost=0.3410
Iteration 229: b0=1.5006, b1=0.9606, cost=0.3406
Iteration 230: b0=1.5015, b1=0.9618, cost=0.3401
Iteration 231: b0=1.5024, b1=0.9631, cost=0.3396
Iteration 232: b0=1.5032, b1=0.9643, cost=0.3392
Iteration 233: b0=1.5040, b1=0.9655, cost=0.3387
Iteration 234: b0=1.5048, b1=0.9668, cost=0.3383
Iteration 235: b0=1.5056, b1=0.9680, cost=0.3379
Iteration 236: b0=1.5064, b1=0.9692, cost=0.3375
Iteration 237: b0=1.5072, b1=0.9704, cost=0.3371
Iteration 238: b0=1.5080, b1=0.9716, cost=0.3367
Iteration 239: b0=1.5087, b1=0.9727, cost=0.3363
Iteration 240: b0=1.5094, b1=0.9739, cost=0.3359
Iteration 241: b0=1.5102, b1=0.9751, cost=0.3355
Iteration 242: b0=1.5109, b1=0.9762, cost=0.3351
Iteration 243: b0=1.5116, b1=0.9774, cost=0.3347
Iteration 244: b0=1.5122, b1=0.9785, cost=0.3344
Iteration 245: b0=1.5129, b1=0.9797, cost=0.3340
Iteration 246: b0=1.5136, b1=0.9808, cost=0.3337
Iteration 247: b0=1.5142, b1=0.9819, cost=0.3333
Iteration 248: b0=1.5148, b1=0.9831, cost=0.3330
Iteration 249: b0=1.5155, b1=0.9842, cost=0.3327
Iteration 250: b0=1.5161, b1=0.9853, cost=0.3324
Iteration 251: b0=1.5167, b1=0.9864, cost=0.3320
Iteration 252: b0=1.5173, b1=0.9875, cost=0.3317
Iteration 253: b0=1.5178, b1=0.9886, cost=0.3314
Iteration 254: b0=1.5184, b1=0.9896, cost=0.3311
Iteration 255: b0=1.5190, b1=0.9907, cost=0.3308
Iteration 256: b0=1.5195, b1=0.9918, cost=0.3305
Iteration 257: b0=1.5200, b1=0.9928, cost=0.3302
Iteration 258: b0=1.5206, b1=0.9939, cost=0.3300
Iteration 259: b0=1.5211, b1=0.9950, cost=0.3297
Iteration 260: b0=1.5216, b1=0.9960, cost=0.3294
Iteration 261: b0=1.5221, b1=0.9970, cost=0.3291
Iteration 262: b0=1.5225, b1=0.9981, cost=0.3289
Iteration 263: b0=1.5230, b1=0.9991, cost=0.3286
Iteration 264: b0=1.5235, b1=1.0001, cost=0.3284
Iteration 265: b0=1.5239, b1=1.0012, cost=0.3281
Iteration 266: b0=1.5244, b1=1.0022, cost=0.3279
Iteration 267: b0=1.5248, b1=1.0032, cost=0.3276
Iteration 268: b0=1.5252, b1=1.0042, cost=0.3274
Iteration 269: b0=1.5257, b1=1.0052, cost=0.3271
Iteration 270: b0=1.5261, b1=1.0062, cost=0.3269
```

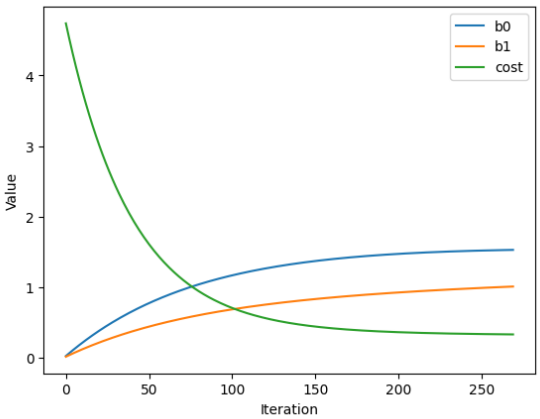
After this, we can also plot the data, the fitting line and the evolution of our parameters:

```python
# Define the predict function
def predict(X, intercept, slope):
    y_pred = intercept + slope * X
    return y_pred

# Compute the predicted values for the regression line
X_line = np.linspace(0, 1, 11)
y_line = predict(X_line, intercept, slope)

# Compute the mean squared error of the model
y_pred = predict(X, intercept, slope)
mse = np.sum((y - y_pred) ** 2) / float(len(y))
```

```
# Plot the data and the regression line
plt.figure(figsize=(8,6))
plt.scatter(X, y, label='Data',color='black')
plt.plot(X_line, y_line, label='Regression Line', color='red')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()

# Define the range of values for slope and intercept
slope_range = np.linspace(slope-0.5, slope+0.5, 100)
intercept_range = np.linspace(intercept-1, intercept+1, 100)
slope_grid, intercept_grid = np.meshgrid(slope_range, intercept_range)

# Compute the cost function for each combination of slope and intercept
cost_grid = np.zeros_like(slope_grid)
for i in range(100):
    for j in range(100):
        y_pred = intercept_range[j] + slope_range[i] * X
        cost_grid[j,i] = np.sum((y - y_pred) ** 2) / float(len(y))

# Plot the contour of the cost function
plt.figure(figsize=(8,6))
plt.contour(slope_grid, intercept_grid, cost_grid, levels=50)
plt.plot(slope, intercept, 'ro')
plt.xlabel('Slope')
plt.ylabel('Intercept')
plt.show()

# Plot the evolution of b0, b1, and the cost function
plt.plot(range(n_iterations), b0_history, label='b0')
plt.plot(range(n_iterations), b1_history, label='b1')
plt.plot(range(n_iterations), cost_history, label='cost')
plt.xlabel('Iteration')
plt.ylabel('Value')
plt.legend()
plt.show()

# Print the slope, intercept, and mean squared error
print(f"Slope: {slope:.4f}")
print(f"Intercept: {intercept:.4f}")
print(f"Mean Squared Error: {mse:.4f}")
```








```
Slope: 1.0062
Intercept: 1.5261
Mean Squared Error: 0.3267
```

As we can see, we now have much more visibility into our data, model and how they behave.

Note that the reason that the data and fitting line in our from-scratch model don't exactly match up with the ones from the previous example is that in both cases the data is randomly produced.

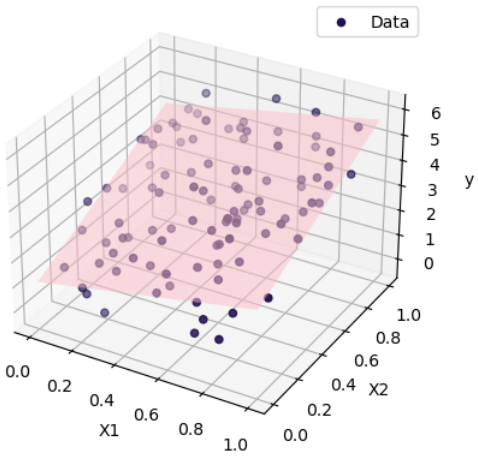
7.6.2 Multiple Regression

The following code performs linear regression on some randomly generated data and visualizes the results in a 3D plot. Here's a breakdown of the different parts of the code:

```
import numpy as np
import matplotlib.pyplot as plt

# Generate some random data
np.random.seed(0)
n_samples = 100
X = np.random.rand(n_samples, 2)
y = 1 + 2*X[:,0] + 3*X[:,1] + np.random.normal(0, 1, n_samples)

# Define the gradient descent function
def gradient_descent(X, y, alpha=0.01, num_iters=100):
    n_samples, n_features = X.shape
    theta = np.zeros(n_features)
    cost_history = np.zeros(num_iters)
    theta_history = np.zeros((num_iters, n_features))

    for i in range(num_iters):
        y_pred = X.dot(theta)
        error = y_pred - y
        theta -= alpha * (1/n_samples) * X.T.dot(error)
        theta_history[i] = theta
        cost_history[i] = (1/n_samples) * np.sum(np.square(error))

    return theta, theta_history, cost_history

# Add a column of ones to X for the intercept term
X = np.c_[np.ones(n_samples), X]

# Set the learning rate and number of iterations
alpha = 0.1
num_iters = 1000

# Run gradient descent
theta, theta_history, cost_history = gradient_descent(X, y, alpha=alpha,
num_iters=num_iters)

# Extract the slope(s) and intercept
slopes = theta[1:]
intercept = theta[0]

# Print the slope(s) and intercept
print(f"Slopes: {slopes}")
print(f"Intercept: {intercept}")

# Predict values for new data
X_new = np.array([[0.2, 0.3], [0.4, 0.5]])
```

```
X_new = np.c_[np.ones(X_new.shape[0]), X_new]
y_new = X_new.dot(theta)

# Print the final cost function value
print(f"Final Cost Function: {cost_history[-1]}")

# Plot the results
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(X[:,1], X[:,2], y, label='Data', cmap='cool')
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('y')
ax.legend()

# Plot the regression plane
xx, yy = np.meshgrid(np.linspace(0, 1, 10), np.linspace(0, 1, 10))
zz = intercept + slopes[0]*xx + slopes[1]*yy
ax.plot_surface(xx, yy, zz, alpha=0.5, color='pink')

plt.show()

# Plot the evolution of the parameters and the cost function

fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot(theta_history)
axs[0].set_title('Evolution of Parameters')
axs[0].set_xlabel('Iteration')
axs[0].set_ylabel('Value')
axs[0].legend(['Slope 1', 'Slope 2', 'Intercept'])
axs[1].plot(cost_history)
axs[1].set_title('Evolution of Cost Function')
axs[1].set_xlabel('Iteration')
axs[1].set_ylabel('Cost')
plt.show()
```

This code outputs the following plot and info:



```
Slopes: [1.57370044 3.02957506]
Intercept: 1.114503255617778
```

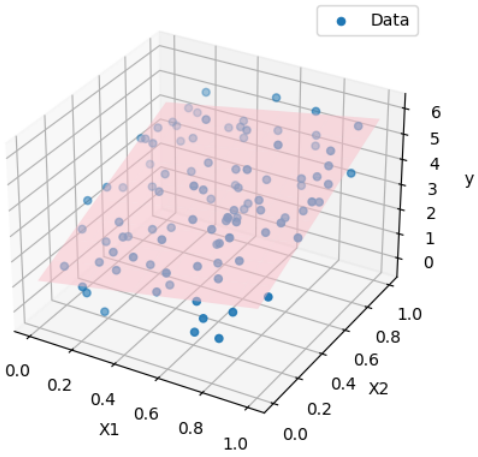
Here's a from-the-ground-up code of the whole thing, with granular control over the number of iterations and access to the evolution of the parameters over each iteration:

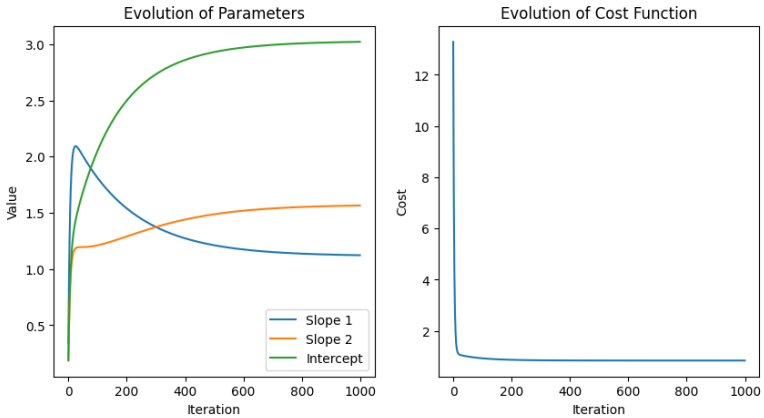
```
import numpy as np
import matplotlib.pyplot as plt

# Generate some random data
np.random.seed(0)
n_samples = 100
X = np.random.rand(n_samples, 2)
y = 2*X[:,0] + 3*X[:,1] + np.random.normal(0, 1, n_samples)

# Define the gradient descent function
def gradient_descent(X, y, alpha=0.01, num_iters=100):
    n_samples, n_features = X.shape
    theta = np.zeros(n_features)
    cost_history = np.zeros(num_iters)
    theta_history = np.zeros((num_iters, n_features))

    for i in range(num_iters):
        y_pred = X.dot(theta)
        error = y_pred - y
        theta -= alpha * (1/n_samples) * X.T.dot(error)
        theta_history[i] = theta
        cost_history[i] = (1/n_samples) * np.sum(np.square(error))

    return theta, theta_history, cost_history

# Run gradient descent
alpha = 0.1
num_iters = 1000
theta, theta_history, cost_history = gradient_descent(X, y, alpha=alpha,
num_iters=num_iters)

# Print the slope and intercept
slope = theta
intercept = 0
print(f"Slope: {slope}")
print(f"Intercept: {intercept}")

# Compute the predicted values for the regression line
X_line = np.array([[0, 0], [1, 1]])
y_line = X_line.dot(theta)

# Print the final cost function value
print(f"Final Cost Function: {cost_history[-1]}")

# Plot the data and the regression line
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
```

```
ax.scatter(X[:,0], X[:,1], y, label='Data', cmap='cool')
ax.plot(X_line[:,0], X_line[:,1], y_line, label='Regression Line', color='red')
ax.set_xlabel('X1')
ax.set_ylabel('X2')
ax.set_zlabel('y')
ax.legend()
plt.show()

# Plot the evolution of the parameters and the cost function
fig, axs = plt.subplots(1, 2, figsize=(10, 5))
axs[0].plot(theta_history)
axs[0].set_title('Evolution of Parameters')
axs[0].set_xlabel('Iteration')
axs[0].set_ylabel('Value')
axs[1].plot(cost_history)
axs[1].set_title('Evolution of Cost Function')
axs[1].set_xlabel('Iteration')
axs[1].set_ylabel('Cost')
plt.show()
```

This outputs the following:

```
Slopes: [1.21047174 2.04635352]
Intercept: 1.8144387153323294
```





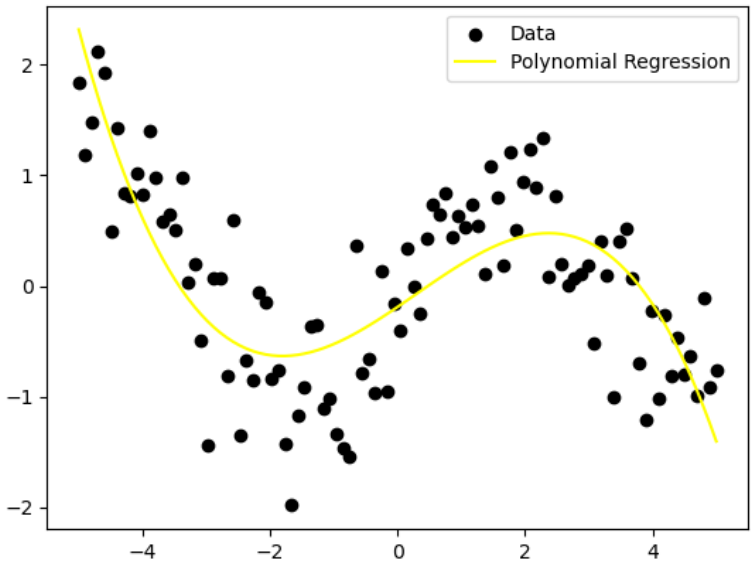
7.6.3 Polynomial regression

Consider the following block of code, with the comments inserted inside the code block instead of breaking it down:

```
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt # Import Matplotlib library for visualization
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Generate some random data
# Set a random seed for reproducibility
np.random.seed(0)
# Generate 100 evenly spaced points between -5 and 5 as input values
x = np.linspace(-5, 5, 100)
# Generate the corresponding output values as the sine of the input plus some noise
y = np.sin(x) + np.random.normal(0, 0.5, 100)

# Reshape the input values to a 2D array for compatibility with Scikit-learn
X = x.reshape(-1, 1)

# Transform the data using polynomial features
# Create an instance of PolynomialFeatures to transform the data
poly = PolynomialFeatures(degree=3)
# Fit and transform the input data to polynomial features of degree 3
X_poly = poly.fit_transform(X)

# Train a linear regression model on the transformed data
model = LinearRegression()
model.fit(X_poly, y)

# Get the model parameters intercept and coefficients
intercept = model.intercept_
multipliers = model.coef_

# Compute the loss
y_pred = model.predict(X_poly)
loss = np.mean((y_pred - y) ** 2)

# Predict values for new data
# Generate 100 evenly spaced points between -5 and 5 as new input values
X_new = np.linspace(-5, 5, 100).reshape(-1, 1)
# Transform the new input data to polynomial features of degree 3
X_new_poly = poly.transform(X_new)
# Predict the output values of the new input data using the trained model
y_new = model.predict(X_new_poly)

# Plot the results
# Plot the original data points
plt.scatter(x, y, label='Data',color='black')
# Plot the predicted output values of the new input data
plt.plot(X_new, y_new, label='Polynomial Regression', color='yellow')
```

```
# Add a legend to the plot
plt.legend()
# Show the plot
plt.show()

# Print model information
print("Intercept:", intercept)
print("Multipliers:", multipliers)
print("Loss:", loss)
```

Here are the outputs:




```
Intercept: -0.19028951840975852
Multipliers: [ 0.          0.39219521  0.02589999 -0.03058996]
Loss: 0.31635512090370743
```

We leave the ground-up coding of this model to the reader as an exercise.

Notes

[43]The formula $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ is commonly used in the context of gradient descent optimization, where the factor of $\frac{1}{2}$ is included for convenience. When this cost function is used, the gradient of the cost function with respect to the parameters becomes simpler, as the factor of 2 cancels out during differentiation. This can make the implementation of gradient descent slightly more efficient. On the other hand, the formula $J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$ is sometimes used outside of gradient descent optimization, where the factor of $\frac{1}{2}$ is not necessary. In fact, this is the more commonly used form of the MSE cost function outside of optimization contexts. Regardless of which formula is used, the basic idea behind the MSE cost function is the same: to measure the difference between the predicted values of the model and the actual values of the target variable, and to minimize this difference by adjusting the parameters of the model

[44]The expression $\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$ is the update rule for the parameters θ in the gradient descent algorithm. It is a general form that applies to different variations of gradient descent, including batch, stochastic, and mini-batch. The difference between these variations lies in the way the gradients are computed and the parameters are updated. In batch gradient descent, the gradient is calculated by summing over all the training examples, while in stochastic gradient descent, the gradient is computed for each training example individually. Mini-batch gradient descent is a hybrid approach that computes the gradient for a subset of the training examples (a mini-batch) at each iteration. In all cases, the update rule is the same, and it is applied to each parameter θ_j.

[45]The image below is taken from https://www.jeremyjordan.me/nn-learning-rate/

[46]At the end of this chapter we will see how we can obtain these exact numbers in Python.

7.7 Further Reading

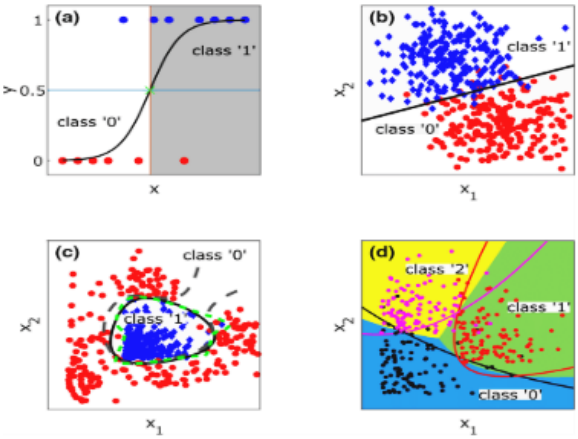
Here are a few references on linear regression:

1. *Introduction to Linear Regression Analysis*, by Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining [25]. This is an excellent resource for anyone looking to gain a deep understanding of linear regression. The book is written in a clear and concise style and covers both the theory and practical applications of linear regression. It includes detailed discussions of model selection, diagnostics, and assumptions, and provides a wealth of examples and exercises to help readers reinforce their understanding.
2. *Linear Regression Analysis*, by George A. F. Seber and Alan J. Lee [35]. This book provides a comprehensive treatment of linear regression, including the theory behind it, as well as practical considerations such as model selection and diagnostics. It covers both simple and multiple regression, and includes discussions of the assumptions underlying the method.
3. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow*, by Sebastian Raschka and Vahid Mirjalili [30]. This book includes a chapter on linear regression in Python, using the scikit-learn library. It covers simple and multiple regression, regularization, and other related topics.

Chapter 8

Logistic Regression

8.1 Introduction



As we learned in the previous chapter, linear regression is a statistical method used to model the relationship between a dependent variable and one or more independent variables. The basic equation for linear regression is:

$$Y = \theta_0 + \theta_1 x + ... + \theta_n$$

However, this method is only suitable for working with *quantitative* data, and it is not appropriate for dealing with qualitative data.

Logistic regression is a statistical method used to analyze a dataset where the dependent variable is qualitative or discrete. In other words, logistic regression is used when we have a dataset where the outcome is either yes or no, 0 or 1, true or false, any of the groups A, B, C or D, and so on. The goal of logistic regression is to find the relationship between the dependent variable and one or more independent variables.[47]

Logistic regression is commonly used in areas such as medical research, marketing, and finance. For example, a medical researcher might use logistic regression to predict the likelihood of a patient having a certain disease based on their age, sex, and other factors; a marketer might use logistic regression to predict the likelihood of a customer buying a product based on their demographic data and previous purchasing behavior; and financial analyst might use logistic regression to predict the likelihood of a borrower defaulting on a loan based on their credit score, income, and other factors.

Logistic regression can serve as a foundation for more advanced modeling techniques. Many more complex models, such as decision trees, random forests, and gradient boosting, are built on top of logistic regression. By understanding the principles behind logistic regression, data scientists can better understand these more complex models and use them more effectively.

8.2 The Logistic Function

As was mentioned, linear regression models are typically used to model the relationship between a continuous dependent variable and one or more independent variables. However, in some cases, the dependent variable may be a binary outcome, such as a 0/1 or true/false response. In such cases, the usual linear regression model may not be appropriate because it assumes that the dependent variable is continuous and can take on any value in the range of real numbers.

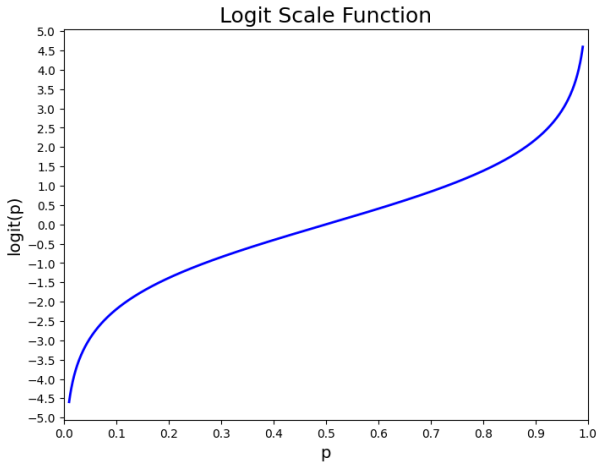
To adapt linear regression methods to a binary outcome, some efforts have focused on transforming the probability values of the binary outcome into a continuous variable that can take on any value in the range of real numbers. One such transformation is the *log-odds* transformation, also known as the ***logit scale***, which maps probabilities to the log-odds of the event occurring, a continuous variable that can

take on any value in the range of real numbers:

In the logit scale, the log-odds of an event occurring is used as the measure of probability, rather than the probability itself. The log-odds is defined as the logarithm of the odds, which is the ratio of the probability of the event occurring to the probability of the event not occurring:

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) \tag{8.1}$$

where p is the probability of an event occurring, and ln is the natural logarithm. The logit function transforms the probability of an event occurring to a continuous range of values between negative and positive infinity.

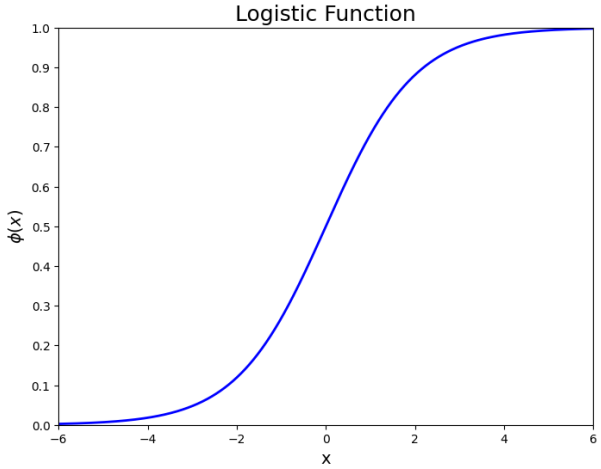


Once the probabilities have been transformed into log-odds, the resulting data can be used in a linear regression model to predict the log-odds of the outcome as a function of one or more predictor variables. However, since the log-odds values can take on any value in the range of real numbers, the resulting model may not be well-suited for predicting probabilities, which are restricted to the range of 0 to 1.

To address this issue, some methods have been proposed to transform the predicted log-odds values back into probabilities using an inverse transformation. One such transformation is the logistic function, which maps log-odds to probabilities between 0 and 1. The ***logistic function*** is defined as:

$$\phi(x) = \frac{1}{1+e^{-z}} \tag{8.2}$$

where e is the base of the natural logarithm, known as Euler's number.



In order to model the relationship between the predictor variables and the outcome variable, the logistic regression model assumes that the relationship between the predictor variables and the outcome variable is linear on the logit scale:

$$p = \frac{1}{1+e^{-(\theta_0+\theta_1 x \ldots+\theta_n)}} \tag{8.3}$$

where p is the probability of an event occurring.

8.3 Estimating the Coefficients

The coefficients $\theta_0, \theta_1, ..., \theta_n$ in the logistic regression equation are unknown and must be estimated based on the available training data. Although we could use the so-called (non-linear) 'least squares' to fit the model, the more general method of ***maximum likelihood*** is preferred since it has better statistical properties.

This method is carried out using the following function, known as the 'likelihood function':

$$L(\theta) = \prod_{i=1}^{n} p(x_i)^{y_i}\,(1-p(x_i))^{(1-y_i)} \tag{8.4}$$

where $\theta = (\theta_0, ..., \theta_n)$, $p(x_i)$ is the predicted probability of the ith observation being in the positive class and y_i is the actual class label (0 or 1) for the ith observation.

To maximize the likelihood function, we take the derivative of the log-likelihood function with respect to the regression coefficients. That is:

$$\frac{\partial}{\partial\theta_j}\ell(\theta) = \sum_{i=1}^{n}\left(y_i - p(x_i)\right)x_{i,j} \tag{8.5}$$

where $\ell(\theta) := \ln L(\theta)$ is the log-likelihood function, and $x_{i,j}$ is the jth predictor variable for the ith observation.

We then use an optimization algorithm such as gradient descent to iteratively update the regression coefficients in the direction of the negative gradient of the log-likelihood function until convergence. Once we have the maximum likelihood estimates of the regression coefficients, we can use them to make predictions on new data.

Example 8.3.1. Suppose we have a dataset of 10 observations, where each observation has two features x_1 and x_2 and a binary outcome (0 or 1). The dataset looks like this:

x_1	x_2	y
1	0	0
2	1	0
1	1	0
3	0	0
3	1	1
4	0	1
5	1	1
6	0	1
6	1	1
7	0	1

Our goal is to use logistic regression to predict the binary outcome y based on the two features x_1 and x_2.

To estimate the coefficients, we can start with some initial values for θ_0, θ_1, and θ_2, and use an optimization algorithm (such as gradient descent) to iteratively update the values of the coefficients until the log-likelihood function is maximized. We use gradient descent here.

The first step is to initialize the coefficients: $\theta_0 = 0$, $\theta_1 = 0$, and $\theta_2 = 0$. Compute the predicted probabilities for each observation using the logistic function:

$$p_i = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x_{1i} + \theta_2 x_{2i})}}, \qquad i = 1, 2, \ldots, 10.$$

Compute the log-likelihood function using the predicted probabilities and the observed binary outcomes:

$$L(\theta_0, \theta_1, \theta_2) = \sum_{i=1}^{10} [y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i)].$$

We then update the coefficients using the gradient descent algorithm:

$$\theta_j = \theta_j - \alpha \frac{\partial L}{\partial \theta_j}, \qquad j = 0, 1, 2,$$

where α is the learning rate and $\frac{\partial L}{\partial \theta_j}$ is the partial derivative of the log-likelihood function with respect to θ_j:

$$\frac{\partial L}{\partial \theta_j} = \sum_{i=1}^{10} (y_i - p_i) x_{ji}, \qquad j = 0, 1, 2.$$

Repeat these steps until convergence (i.e., until the change in the log-likelihood function is small or the maximum number of iterations is reached).

Using the partial derivatives, we can update the coefficients using the learning rate $\alpha = 0.1$. After several iterations, we can converge to the maximum likelihood estimates of $\theta_0 = -1.437, \theta_1 = 0.526$, and $\theta_2 = 1.251$. These values give us a logistic regression model that can be used to predict the binary outcome y given new values of x_1 and x_2.

8.4 Evaluating Models

There are several methods for evaluating the performance of a logistic regression model. Here are some commonly used evaluation metrics, most of which are packed into the 'confusion matrix'. We learned about confusion matrices in Chapter 6. Here's a quick refresher:

A ***confusion matrix*** is a table that is used to evaluate the performance of a machine-learning model by comparing the actual and predicted values. It is particularly useful for classification problems, where the objective is to classify instances into one of the several predefined classes.

The confusion matrix has a table layout with two rows and two columns. Each row represents the instances in a predicted class, while each column represents the instances in an actual class. The diagonal of the matrix represents the correctly classified instances, while the off-diagonal elements represent the incorrectly classified instances.

Here is an example of a confusion matrix:

	PP	**PN**
AP	50	10
AN	5	85

where 'PP' stands for Predicted Positive, 'PN' for Predicted Negative, 'AP' for Actual Positive, and 'AN' for Actual Negative.

In this example, there are two classes, Positive and Negative. The rows represent the instances that were predicted to belong to the Positive and Negative classes, while the columns represent the instances that actually belong to those classes.

The matrix shows that out of 60 instances that actually belong to the Positive class, the model correctly classified 50 as Positive, but misclassified 10 as Negative. Similarly, out of 90 instances that actually belong to the Negative class, the model correctly

classified 85 as Negative, but misclassified 5 as Positive.

Based on the confusion matrix, we can calculate several evaluation metrics such as accuracy, precision, recall, and F1-score.

Accuracy is the proportion of all correctly classified instances to the total number of instances in the dataset, and is given by:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{8.6}$$

where TP is the number of true positives, TN is the number of true negatives, FP is the number of false positives, and FN is the number of false negatives.

Precision is the proportion of true positive instances to the total number of instances predicted as positive, and is given by:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{8.7}$$

Recall (also known as *sensitivity* or *true positive rate*) is the proportion of true positive instances to the total number of instances that actually belong to the positive class, and is given by:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{8.8}$$

F1-score is a weighted average of precision and recall, and is given by:

$$\text{F1-score} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{8.9}$$

These metrics provide a comprehensive understanding of the performance of a classification model, and can be used to compare the performance of different models.

8.5 Python Implementations

Here I'll give a full example of a Python implementation of a Logistic Regression Problem. We'll be using the `pima-indians-diabetes` dataset, which can be downloaded from the UCI Machine Learning Repository (you can find this dataset at kaggle.com as well).

The `pima-indians-diabetes` dataset is a popular dataset in machine learning and predictive analytics. The dataset consists of medical data from Pima Indians women aged 21 years and older living near Phoenix, Arizona, USA. The dataset was originally collected by the National Institute of Diabetes and Digestive and Kidney Diseases in the late 1980s.

The dataset includes 768 instances or observations, each with 8 variables. The variables are:

1. Pregnancies: Number of times pregnant
2. Glucose: Plasma glucose concentration 2 hours in an oral glucose tolerance test
3. BloodPressure: Blood pressure (mm Hg)
4. SkinThickness: Triceps skin thickness (mm)
5. Insulin: 2-hour serum insulin (mu U/ml)
6. BMI: Body mass index (weight in kg/height)
7. DiabetesPedigreeFunction: Diabetes pedigree function (a function that estimates the likelihood of diabetes based on family history)
8. Age: Age (years)

The target variable is a binary variable indicating whether or not the patient has diabetes. A value of 1 indicates that the patient has diabetes, while a value of 0 indicates that the patient does not have diabetes.

The Pima Indians Diabetes Dataset is often used as a benchmark dataset for evaluating and comparing different machine learning algorithms for classification, such as logistic regression, decision trees, and support vector machines. The goal is to predict whether or not a patient has diabetes based on their medical data.

Alright, the code in its entirety is as follows:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report

# Load and the dataset
url =
"raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
```

```
data = pd.read_csv(url, names=names)
print(data)


# Split the data into input and output variables
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Create a logistic regression object and fit the model
model = LogisticRegression(max_iter=1000)
model.fit(X, y)

# Print the coefficients
print("Coefficients: ", model.coef_)

# Make predictions
y_pred = model.predict(X)

# Evaluate the model
print('')
print("Confusion Matrix:\n", confusion_matrix(y, y_pred))
print("Classification Report:\n", classification_report(y, y_pred))
```

This code outputs the following results:

```
     preg  plas  pres  skin  test  mass   pedi  age  class
0       6   148    72    35     0  33.6  0.627   50      1
1       1    85    66    29     0  26.6  0.351   31      0
2       8   183    64     0     0  23.3  0.672   32      1
3       1    89    66    23    94  28.1  0.167   21      0
4       0   137    40    35   168  43.1  2.288   33      1
..    ...   ...   ...   ...   ...   ...    ...  ...    ...
763    10   101    76    48   180  32.9  0.171   63      0
764     2   122    70    27     0  36.8  0.340   27      0
765     5   121    72    23   112  26.2  0.245   30      0
766     1   126    60     0     0  30.1  0.349   47      1
767     1    93    70    31     0  30.4  0.315   23      0

[768 rows x 9 columns]
Coefficients:  [[ 1.22499378e-01  3.51098443e-02 -1.32990710e-02  7.81696950e-04
  -1.17387556e-03  8.96500885e-02  8.67909361e-01  1.49853525e-02]]

Confusion Matrix:
 [[444  56]
 [112 156]]
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       500
           1       0.74      0.58      0.65       268

    accuracy                           0.78       768
   macro avg       0.77      0.74      0.75       768
weighted avg       0.78      0.78      0.77       768
```

A ground-up version of this code snippet can be given as follows, where don't use `Sklearn` at all, and we use gradient descent as our optimizing method:[48]

```
import pandas as pd
import numpy as np

# Define the sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Load and preprocess the dataset
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
data = pd.read_csv(url, names=names)

print(data,'\n')

# Split the data into input and output variables
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Add a column of ones to the input variables
X = np.hstack((np.ones((X.shape[0], 1)), X))

# Initialize the weights
w = np.zeros(X.shape[1])

# Set the learning rate and number of iterations
learning_rate = 0.1
num_iterations = 5000

# Implement the gradient descent algorithm
for i in range(num_iterations):
    z = np.dot(X, w)
    y_pred = sigmoid(z)
    error = y_pred - y
    gradient = np.dot(X.T, error) / y.size
    w -= learning_rate * gradient

# Make predictions
y_pred = np.round(sigmoid(np.dot(X, w)))

# Evaluate the model
confusion_matrix = np.zeros((2, 2), dtype=int)
for i in range(y.size):
    if y[i] == 1 and y_pred[i] == 1:
        confusion_matrix[1, 1] += 1
    elif y[i] == 1 and y_pred[i] == 0:
        confusion_matrix[1, 0] += 1
    elif y[i] == 0 and y_pred[i] == 1:
        confusion_matrix[0, 1] += 1
    else:
```

```
        confusion_matrix[0, 0] += 1

accuracy = (confusion_matrix[0, 0] + confusion_matrix[1, 1]) / y.size
precision = confusion_matrix[1, 1] / (confusion_matrix[1, 1] + confusion_matrix[0, 1])
recall = confusion_matrix[1, 1] / (confusion_matrix[1, 1] + confusion_matrix[1, 0])
f1_score = 2 * precision * recall / (precision + recall)

print("Confusion Matrix:\n", confusion_matrix)
print("\nAccuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1_score))
```

This outputs the following results, which aren't impressive in comparison to what we got from Sklern's built-in logistic model:

```
     preg  plas  pres  skin  test  mass   pedi  age  class
0       6   148    72    35     0  33.6  0.627   50      1
1       1    85    66    29     0  26.6  0.351   31      0
2       8   183    64     0     0  23.3  0.672   32      1
3       1    89    66    23    94  28.1  0.167   21      0
4       0   137    40    35   168  43.1  2.288   33      1
..    ...   ...   ...   ...   ...   ...    ...  ...    ...
763    10   101    76    48   180  32.9  0.171   63      0
764     2   122    70    27     0  36.8  0.340   27      0
765     5   121    72    23   112  26.2  0.245   30      0
766     1   126    60     0     0  30.1  0.349   47      1
767     1    93    70    31     0  30.4  0.315   23      0

[768 rows x 9 columns]

Confusion Matrix:
 [[230 270]
 [ 46 222]]

Accuracy: 0.59
Precision: 0.45
Recall: 0.83
F1 Score: 0.58
```

Notes

[47]The image above is taken from this article: https://www.researchgate.net/publication/353913155_Mitigation_of_nonlinear_phase_noise_in_single-channel_coherent_16-QAM_systems_employing_logistic_regression

[48]Logistic Regression supports only optimizers in ['liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga']. To specify the optimizer in your model, e.g., 'sag', you need to add the parameter `solver='sag'` in the model variable. By default, the solver parameter is set to 'lbfgs', which is a quasi-Newton method.

8.6 Further Reading

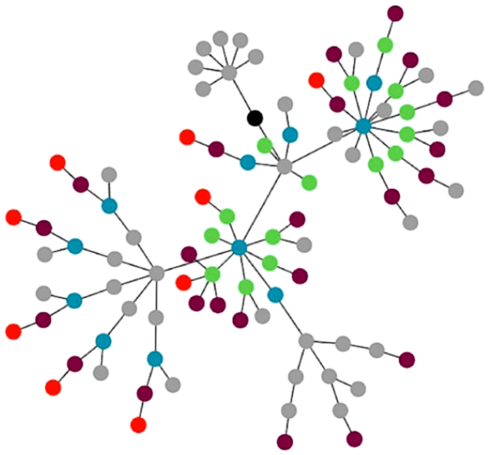
Here are a few useful resources on logistic regression:

1. *Logistic Regression: A Self-Learning Text*, by David G. Kleinbaum and Mitchel Klein [19]. This textbook provides a comprehensive treatment of logistic regression analysis, including the concepts of maximum likelihood estimation, model fit, model selection, and inference. It also includes practical examples and exercises for the application of logistic regression analysis in various fields.
2. *Applied Logistic Regression*, by David W. Hosmer, Jr., Stanley Lemeshow, and Rodney X. Sturdivant [22]. This textbook provides a practical approach to logistic regression analysis, including the concepts of model build-

ing, model assessment, and model selection. It includes numerous examples and case studies from various fields, as well as computer code and data sets for the application of logistic regression analysis.

Chapter 9

Tree Methods



9.1 Introduction

Tree methods are a powerful class of machine learning algorithms that can handle both categorical and numerical data, are easy to interpret and can be used for both classification and regression tasks. These algorithms build decision trees that model decisions and their possible consequences. Each node in the tree represents a decision, while the branches represent possible outcomes of that decision.[49]

Decision trees, random forests, gradient boosting, and XGBoost are some popular examples of tree-based methods that can be used to solve a wide range of machine learning problems. In this section we'll introduce these and their underlying mathematics.

9.2 Decision Trees

A ***decision tree*** is a tree-like model where each internal node represents a decision based on the value of a feature, and each leaf node represents a final decision about the target variable. The model is built by recursively partitioning the data based on the values of the features, with the goal of minimizing the impurity of the resulting subsets.

Entropy is a measure of the impurity of a set of samples. It is commonly used as a criterion for evaluating the quality of splits in decision trees. The formula for entropy is this:

$$H(S) = -\sum_{i=1}^{c} p_i \log_2(p_i) \tag{9.1}$$

where S is the set of samples, c is the number of classes, and p_i is the proportion of samples in S that belong to class i. Entropy is minimized when all samples belong to the same class and is maximized when the samples are evenly split between classes.

Information gain is a measure of how much entropy is reduced by splitting the data based on a particular feature. The formula for information gain is:

$$IG(S, F) = H(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} H(S_v) \tag{9.2}$$

where S is the set of samples, F is the feature being considered, $Values(F)$ is the set of possible values for feature F, $|S_v|$ is the number of samples in S that have value v for feature F, and $H(S_v)$ is the entropy of the subset of samples in S that have value v for feature F. Information gain is maximized when splitting the data based on a particular feature results in the greatest reduction in entropy.

In decision tree algorithms, the goal is to split the data based on features that best separate the target variable (or class label) into the purest subsets possible at each node.

When building a decision tree, all features are considered for splitting at each node. The algorithm searches through all possible splits for each feature to find the best one that minimizes the impurity of the resulting subsets. The feature with the highest

information gain or reduction in impurity is selected for the split.

However, including all features can lead to overfitting, where the model becomes too complex and is fit too closely to the training data. To prevent overfitting, we can use techniques such as pruning or feature selection to simplify the tree.

To determine the threshold for a given feature, the algorithm searches through all possible split points (or thresholds) for that feature and selects the one that results in the greatest information gain or reduction in impurity. For example, if the feature is continuous, we might try splitting it at different values and see which one results in the greatest reduction in impurity. If the feature is categorical, we might try splitting it on each category and see which one results in the greatest reduction in impurity.

Ultimately, the choice of which feature to split on and what threshold to use depends on the data and the specific problem being solved. The decision tree algorithm will search through all possible options and select the best one based on the chosen criterion.

More specifically, here's what goes into the workings of the decision tree algorithm:

1. It starts with the entire dataset and calculates the entropy of the target variable. It then calculates the information gain of splitting the data based on each feature and chooses the feature that results in the highest information gain. Remember that information gain is a measure of the reduction in uncertainty (or entropy) achieved by splitting the data based on a particular feature. The higher the information gain, the more effective the split is in reducing the entropy and separating the classes in the data.
2. It then splits the data based on the feature with the highest information gain and repeats the process on each resulting subset until a stopping criterion is met, such as reaching a maximum depth or having a minimum number of samples in each leaf node.

When we split the data based on the feature with the highest information gain, we are essentially dividing the dataset into two or more subsets that are more homogenous in terms of the target variable. This division enables us to create a simpler set of decision rules that can be used to classify new data points. The idea is that by recursively splitting the data based on the feature with the highest information gain, we can create a tree structure that accurately represents the underlying decision-making process of the data.

Choosing the feature with the highest information gain at each split is known as the "greedy" strategy. It is called greedy because it does not look ahead to see if the split will lead to a better overall decision tree structure. However, this strategy often leads to reasonably good decision trees, especially when combined with pruning techniques that remove unnecessary branches in the tree.

We now do an example to illustrate everything that was said above:

Example 9.2.1. Suppose we have a dataset of animals with features such as weight, height, and type of food. The target variable is whether the animal is a herbivore or a carnivore. We can use entropy and information gain to build a decision tree to predict the diet of an animal based on its features.

$$\begin{cases}\text{Features: weight, height}\\ \text{Target: herbivore, carnivore}\end{cases}$$

First, we calculate the entropy of the target variable. Suppose there are 100 animals in the dataset, with 70 herbivores and 30 carnivores. The entropy is:

$$H(S) = -\left(\frac{70}{100}\log_2\frac{70}{100} + \frac{30}{100}\log_2\frac{30}{100}\right) = 0.881$$

Next, we calculate the information gain for each feature. Suppose weight is the first feature. As one of the many possible choices of threshold (that the algorithm actually goes through) we split the dataset based on whether the weight is greater than or equal to 50 kg. The subsets are:

- Subset 1: animals with weight < 50 kg, with 30 herbivores and 20 carnivores
- Subset 2: animals with weight >= 50 kg, with 40 herbivores and 10 carnivores

We calculate the entropy of each subset:

$$H(S_1) = -\left(\frac{30}{50}\log_2\frac{30}{50} + \frac{20}{50}\log_2\frac{20}{50}\right) = 0.971$$
$$H(S_2) = -\left(\frac{40}{50}\log_2\frac{40}{50} + \frac{10}{50}\log_2\frac{10}{50}\right) = 0.722$$

Then, we calculate the information gain of splitting on weight:

$$\begin{aligned} IG(S, weight) &= H(S) - \frac{50}{100}H(S_1) - \frac{50}{100}H(S_2)\\ &= 0.881 - \frac{50}{100}0.971 - \frac{50}{100}0.722 = 0.119\end{aligned}$$

Next, suppose height is the second feature. We split the dataset based on whether the height is greater

than or equal to 1 meter (again, as a sample threshold that the algorithm will likely go through and assess, in actuality). The subsets are:

- Subset 1: animals with height < 1 meter, with 40 herbivores and 10 carnivores
- Subset 2: animals with height >= 1 meter, with 30 herbivores and 20 carnivores

We calculate the entropy of each subset:

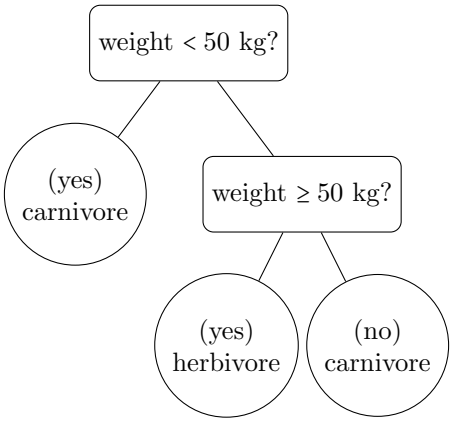
$$H(S_1) = -\left(\frac{40}{50}\log_2\frac{40}{50} + \frac{10}{50}\log_2\frac{10}{50}\right) = 0.722$$

$$H(S_2) = -\left(\frac{30}{50}\log_2\frac{30}{50} + \frac{20}{50}\log_2\frac{20}{50}\right) = 0.971$$

Then, we calculate the information gain of splitting on height:

$$IG(S, height) = H(S) - \frac{50}{100}H(S_1) - \frac{50}{100}H(S_2)$$
$$= 0.881 - \frac{50}{100}0.722 - \frac{50}{100}0.971 = 0.019$$

Weight has a higher information gain, so we split the dataset based on weight. The resulting decision tree might look something like this (or really, something quite different due to many other similar calculations that the algorithm performs and compares to one another):



In this example tree, the decision tree predicts that an animal is a carnivore if its weight is less than 50 kg, and otherwise predicts the animal's diet based on whether its weight is greater than or equal to 50 kg. This decision tree was built using entropy and information gain to determine the best splits at each node.

Example 9.2.2. Suppose you are a farmer and want to determine which crop to plant based on the weather conditions. You have three options: wheat, corn, and soybeans. You have collected data on the weather conditions and the resulting crop yield for the past five years. Your data set looks like this:

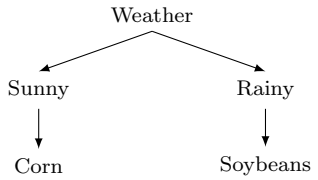
Weather	Crop	Yield
Sunny	Wheat	2
Rainy	Wheat	1
Sunny	Corn	4
Rainy	Corn	3
Sunny	Soybean	3
Rainy	Soybean	2
Sunny	Soybean	4
Rainy	Soybean	1
Sunny	Wheat	3
Rainy	Corn	2

To create a decision tree to determine which crop to plant based on the weather, we first calculate the entropy of the data set, as follows:

- There are 5 sunny days and 5 rainy days in the data set, so the probability of a sunny day is 0.5 and the probability of a rainy day is 0.5.
- For each weather condition, we calculate the proportion of each crop that was planted and the resulting yield. For example, on sunny days, 40% of the crops planted were wheat, with an average yield of 2.5; 20% were corn, with an average yield of 4; and 40% were soybeans, with an average yield of 3.5.
- We then calculate the entropy of each crop for each weather condition using the formula: entropy = $-p\log_2(p_i)) - (1-p)\log_2(1-p_i)$, where p is the proportion of each crop. For example, the entropy of wheat on sunny days is calculated as follows: entropy(sunny) = $-0.4\log_2(0.4) - 0.6\log_2(0.6) = 0.971$.
- We then calculate the entropy of each crop for each weather condition and multiply it by the proportion of each weather condition. For example, the entropy of wheat is 0.971 on sunny days and 0.971 on rainy days, so the weighted entropy of wheat is 0.5 × 0.971 + 0.5 × 0.971 = 0.971.

We then calculate the information gain of each crop by subtracting the weighted entropy of the crop from the overall entropy of the data set. For example, the information gain of wheat is 0.985 - 0.971 = 0.014. The crop with the highest information gain is chosen as the first node of the decision tree.

In the end, the decision tree might look something like this:



9.3 Ensembling Trees

Ensemble methods are popular approaches that combine numerous simple 'building block' models to create a singular, robust model with enhanced predictive power. These fundamental models are typically referred to as 'weak learners', as they often produce sub-optimal predictions in isolation.[50]

In this section, we will explore several well-known ensemble methods, including bagging, random forests, boosting, and Bayesian learners additive regression trees. These methods employ regression or classification trees as their basic building blocks. By combining numerous trees, each with a unique perspective on the data, these ensemble methods can leverage the strength of multiple models to overcome the limitations of any one model, ultimately leading to more accurate and reliable predictions.

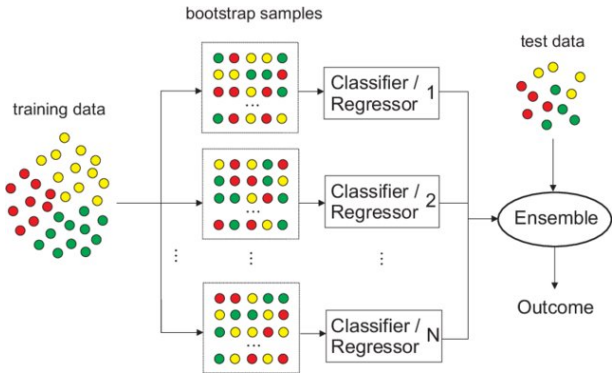
9.3.1 Bagging

Bagging, or ***bootstrap aggregation***, is a technique used in machine learning to improve the performance of a single model by combining the predictions of multiple models. In bagging, several base models are trained on different subsets of the training data, and their predictions are combined through an aggregation process.

Bagging can be applied to a variety of models, including decision trees, linear models, and neural networks. It is a powerful technique for improving the accuracy and stability of machine learning models, especially in cases where the individual models suffer from high variance.

One of the key assumptions of bagging is that all the base models that compose the ensemble have the same architecture, which results in the same topology, number of input-output variables, and number of parameters to train. This means that the base models should be identical in terms of their structure and the number of parameters they have to learn.

In the context of trees, consider a set of decision trees trained with the bagging technique. In this case, all the trees in the ensemble would have the same branches, with the same number of parameters to train and the same input-output variables. This ensures that each tree in the ensemble learns the same patterns and produces similar predictions, which are then combined through the aggregation process to improve the overall performance of the model.



Let X be the training set of size n, and let $X_1, X_2, \ldots, X_k$ be k bootstrap samples of X. Bagging ensemble is defined based on the nature of the problems it is meant to solve: it is the majority of votes for classification problems, and averaging for regression problems:

$$\begin{cases} f(X) = \arg\max_l \sum_{i=1}^{k} [T_i(X) = l] & \text{Classification} \\ f(X) = \frac{1}{k} \sum_{i=1}^{k} T_i(X) & \text{Regression} \end{cases} \tag{9.3}$$

where k is the number of decision trees, $T_i(X)$ is the prediction made by the ith decision tree for input X and $T_i(X) = l$ is an indicator function that takes the value 1 if the predicted label of the i-th decision tree is l, and 0 otherwise.

The formula calculates the sum of the indicator functions for each possible label l, and returns the label l for which the sum is maximum. In other words, it returns the label that is predicted by the majority of the decision trees in the forest. This formula is equivalent to the previous one, which calculates the mode of the predicted labels.

Example 9.3.1. Suppose we want to predict the price of a house based on its size, location, and age. We have a dataset of 1000 houses and we want to train a bagged model to predict the price. We create 10 bootstrap samples of the data and train a decision tree (or a linear regression model) on each sample. We then average the predictions of the 10 models to obtain the final prediction. The bagged model is more robust than any individual model, as it is less sensitive to the particular data points in the training set.

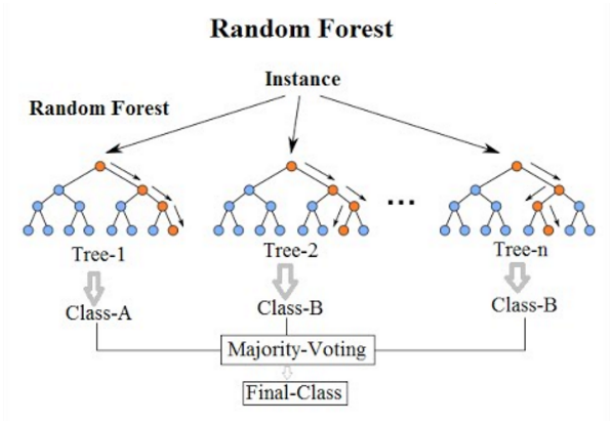
Note that by enforcing this assumption of homogeneity, bagging can help reduce overfitting and improve the generalization performance of the model. However, it also limits the diversity of the base models in the ensemble, which can sometimes result in a suboptimal solution. Therefore, other ensemble

techniques such as random forests and boosting have been developed to address this issue and provide better performance in certain scenarios.

9.3.2 Random Forests

Random forests are one of the most popular and well-known bagging techniques. They rely on decision or regression trees as the base learners but differ from pure bagging techniques in that the topology of the trees is not fixed for all trees in the ensemble.

The idea behind random forests is quite simple: we repeatedly select data from the dataset with replacement and build a decision tree with each new sample. It is worth noting that since we are sampling with replacement, many data points will be repeated and many won't be included at all. This has an impact on measuring the error of a random forest. Another key feature of the random forest is that each node of the decision tree is restricted to only consider splits on random subsets of the features.[51]



Random forests are mathematically defined just like bootstrap aggregation: it takes the majority of votes for classification problems, and averages the votes for regression problems (see (9.3)).

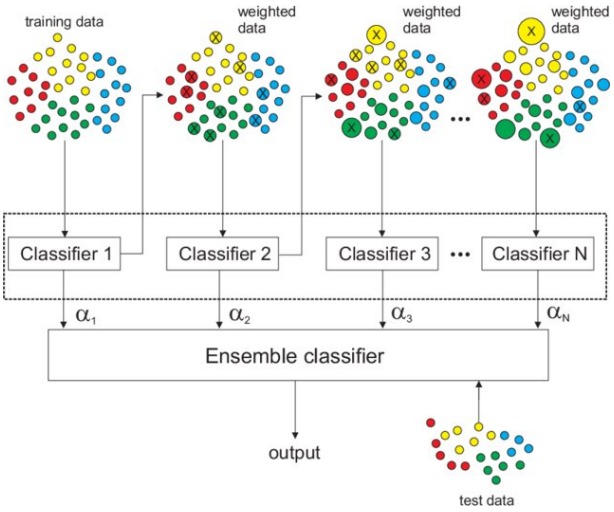
Note, however, that despite the similarity of their formulas, bagging and random forests aren't the same. Bagging creates multiple models trained on different subsets of the training data, while random forests are an extension of bagging that adds *further* randomness by randomly selecting features at each split of the decision tree. In other words, instead of using all features to split a node (as in bagging), in random forests only a random subset of features is considered. This introduces further diversity among the base models and helps to reduce the correlation between them.

To evaluate the performance of a random forest, we can use a standard cross-validation method by splitting the data into a training and testing set and comparing the predictions to the actual values. However, each tree in the random forest does not get a chance to see all of the training data. As a result, we can use the unseen data to cross-validate each tree individually.

9.3.3 Boosting

Boosting is another ensemble method that combines multiple weak models into a strong model. The goal of boosting is to iteratively train a sequence of models, each of which is focused on the examples that were misclassified by the previous models. This approach can lead to a highly accurate model, even if the individual models are quite simple.

In other words, in each iteration, the Boosting algorithm identifies miss-classified data points, increasing their weights so that the next classifier will pay extra attention to get them right.



Let X be the training set of size n, and let $y \in \{-1, 1\}$ be the binary target variable. The boosting algorithm starts by training a weak learner $f_1(x)$ on the data. The weak learner can be any model that performs better than random guessing, such as a decision tree of limited depth. The algorithm then computes the weighted error of the weak learner:

$$err_1 = \sum_{i=1}^{n} w_i^{(1)} I(y_i \neq f_1(x_i)) \tag{9.4}$$

where $w_i^{(1)}$ is the weight assigned to the ith example, initialized to $w_i^{(1)} = \frac{1}{n}$, and I is the indicator function.

The ***indicator function*** $I(y_i \neq f_1(x_i))$ in the equation for err_1 is a function that evaluates to 1 if $y_i \neq f_1(x_i)$ and 0 otherwise. In other words, it is an indicator of whether the weak learner $f_1(x_i)$ misclassified the ith training example. The weighted error

err_1 is simply the sum of these indicators, where the weights $w_i^{(1)}$ give more weight to examples that are misclassified.

The weights are then updated based on the error of the weak learner:

$$\alpha_1 = \frac{1}{2} \ln \frac{1 - err_1}{err_1} \tag{9.5}$$

$$w_i^{(2)} = \frac{w_i^{(1)} \exp(-\alpha_1 y_i f_1(x_i))}{Z_1} \tag{9.6}$$

where α_1 is the weight assigned to the weak learner, and Z_1 is a normalization factor that ensures that the weights sum to 1.

The algorithm then trains a second weak learner $f_2(x)$ on the data, with the weights $w_i^{(2)}$. Specifically, the second weak learner $f_2(x)$ is trained on the same training set X with the updated weights $w_i^{(2)}$ as follows:

1. Train the weak learner $f_2(x)$ on the training set X with weights $w_i^{(2)}$:

$$f_2(x) = \arg\min_f \sum_{i=1}^{n} w_i^{(2)} I(y_i \neq f(x_i)) \tag{9.7}$$

   where the notation 'arg min $_f$' indicates that we are looking for the hypothesis f that minimizes the weighted error on the training set X, where the weights $w_i^{(2)}$ emphasize the examples that were misclassified by the previous weak learner.

2. Compute the weight α_2 assigned to $f_2(x)$:

$$\alpha_2 = \frac{1}{2} \ln \frac{1 - err_2}{err_2} \tag{9.8}$$

3. Compute the weighted error of $f_2(x)$:

$$err_2 = \sum_{i=1}^{n} w_i^{(2)} I(y_i \neq f_2(x_i)) \tag{9.9}$$

4. Update the weights:

$$w_i^{(3)} = \frac{w_i^{(2)} \exp(-\alpha_2 y_i f_2(x_i))}{Z_2} \tag{9.10}$$

   where Z_2 is a normalization factor that ensures the sum of the weights is 1.

The boosting algorithm then repeats this process, training additional weak learners $f_3(x), f_4(x), ..., f_T(x)$ on the updated data sets, where T is the number of iterations (or 'boosting rounds') specified by the user. Each weak learner is trained to emphasize the examples that were misclassified by the previous weak learners, and the final prediction is a weighted sum of the weak learners' predictions, where the weights are proportional to the performance of each weak learner.

Boosting can be applied to a variety of models, including decision trees, linear models, and neural networks. It is a powerful technique for improving the accuracy of machine learning models, especially in cases where the individual models suffer from high bias.

9.4 Python Implementations

Here is a Python code that demonstrates how to use the scikit-learn library to build a decision tree classifier and evaluate its accuracy on a test dataset. The example is based on the *iris* dataset. This dataset is a well-known and commonly used dataset in machine learning and statistics. It was introduced by the British statistician and biologist Ronald Fisher in his 1936 paper 'The use of multiple measurements in taxonomic problems' and contains measurements of the sepal length, sepal width, petal length, and petal width for 150 iris flowers belonging to three different species: setosa, versicolor, and virginica. Each flower in the dataset is labeled with its corresponding species.

The iris dataset is often used as a toy dataset for classification tasks and is particularly useful for learning and demonstrating data visualization and pattern recognition techniques. Its popularity stems from the fact that it is a relatively small and simple dataset, yet it exhibits many of the challenges and complexities commonly encountered in real-world datasets.

Different steps of our code are commented on in the code box of higher readability.

```
# Import necessary libraries
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
```

```
iris = datasets.load_iris()

# Split data into training and testing sets
X_train, X_test, y_train, y_test =  train_test_split(iris.data, iris.target,
test_size=0.3)

# Initialize the classifier
clf = DecisionTreeClassifier()

# Train the classifier on the training  data
clf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = clf.predict(X_test)

# Compute the accuracy of the  classifier

accuracy = accuracy_score(y_test,  y_pred)

print("Accuracy:", accuracy)
```

iris data and the target classifications can be summoned by the following commands (where `pd` is used after importing the pandas library using the command `import pandas as pd`):

```
pd.DataFrame(iris['data'])
```

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

```
pd.DataFrame(iris['target'])
```

	0
0	0
1	0
2	0
3	0
4	0
...	...
145	2
146	2
147	2
148	2
149	2

150 rows × 1 columns

Notes

[49]This tree diagram is adapted from https://www.researchgate.net/publication/304374851_A_novel_social_contact_graph-based_routing_strategy_for_workload_and_throughput_fairness_in_delay_tolerant_networks

[50]The images in this section are all either taken from Wikipedia or this paper: https://arxiv.org/abs/2207.07580

[51]The following image is taken from https://en.wikipedia.org/wiki/Random_forest

9.5 Further Readings

Here are some references on tree methods, their foundations and applications.

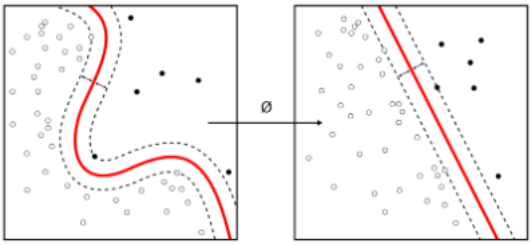
1. *The Elements of Statistical Learning*, by Trevor Hastie, Robert Tibshirani, and Jerome Friedman [16]. This book covers decision trees, bagging, boosting, and random forests in depth.
2. *Applied Predictive Modeling*, by Max Kuhn and Kjell Johnson [21]. This book covers decision

trees, bagging, boosting, and random forests, as well as other predictive modeling techniques.

3. "Random Forests", by Leo Breiman [7]. This is a seminal paper on the random forest algorithm.
4. "Gradient Boosting Machines: A Tutorial", by Alexey Natekin and Alois Knoll [26]. This paper provides a tutorial on gradient boosting, a popular boosting algorithm.
5. "Random Forests for Classification and Regression", by Andy Liaw and Matthew Wiener [23]. This paper provides an overview of the random forest algorithm and its use in classification and regression problems, as well as its implementations in R.
6. *Data Mining: Concepts and Techniques*, by Jiawei Han, Micheline Kamber, and Jian Pei [14]. This book covers decision trees, bagging, boosting, and random forests, as well as other data mining techniques.

Chapter 10

Support Vector Machines



10.1 Introduction

A ***support vector machine*** (***SVM***) is a powerful and popular supervised machine learning algorithm used for classification and regression tasks. SVMs employ kernel methods to implicitly map data to a usually higher-dimensional space. The task, say, of classification, is then performed in this higher dimensional space.[52]

At a high level, the goal of an SVM is to find the best 'hyperplane' (i.e., a line, plane, or hyperplane; more on this below) that separates the data into different classes while maximizing the margin between the hyperplane and the closest data points from each class, called 'support vectors'. The 'margin' is the distance between the hyperplane and the closest data points from each class; a larger margin means that the decision boundary is further away from the data points, reducing the chances of misclassification on new, unseen data.

10.2 Hyperplanes and Margines

In SVM, a ***hyperplane*** is a decision boundary that separates the input data into two classes. In general, for a binary classification problem, a hyperplane is a $(d-1)$-dimensional subspace of the d-dimensional feature space. In the case of a two-class classification problem, the hyperplane is a line that separates the two classes, while in the case of a multi-class classification problem, the hyperplane is a hyperplane that separates the classes in a higher-dimensional space.

Mathematically, a hyperplane is defined as the set of points x that satisfy the equation:

$$w^T x + b = 0 \tag{10.1}$$

where w is the weight vector and b is the bias term. The weight vector determines the orientation of the hyperplane and the bias term determines its position in the n-dimensional space.

The SVM algorithm seeks to find the optimal hyperplane that maximizes the margin between the two classes.

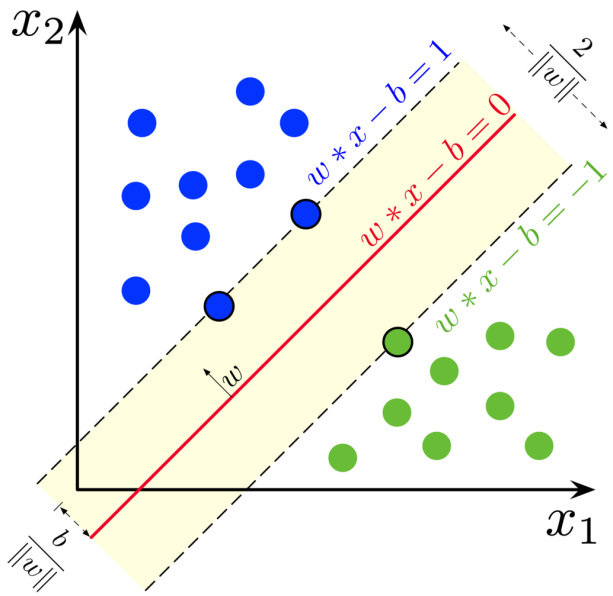
If the training data is linearly separable, we can select two parallel hyperplanes that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the ***hard margin***, and the maximum-margin hyperplane is the hyperplane that lies halfway between them.

More specifically, hard margin is defined as the perpendicular distance between the hyperplane and the closest data point from either class. Let x_i be a data point closest to the hyperplane, and let $y_i \in \{-1, 1\}$ be its corresponding class label. Then, under the assumption of normalized data the margin can be calculated as follows:[53]

$$\text{margin} = y_i(w^T x_i + b) \tag{10.2}$$

We also have to prevent data points from falling into the margin, we add the following constraint:

$$y_i(w^T x_i - b) \geq 1, \forall 1 \leq i \leq n \tag{10.3}$$



We can put this together to get the optimization problem:

$$\begin{aligned} &\text{minimize } (\frac{1}{2}\|w\|^2) \\ &\text{Subject to:} \\ &y_i(w^T x_i + b) \geq 1 \ \forall i \end{aligned} \tag{10.4}$$

A hard margin SVM does not allow any misclassification of the training data points. However, in many real-world scenarios, this is not possible because of noise or overlapping data points. In such cases, a soft margin SVM is used.

A ***soft margin*** SVM allows for some misclassification of the training data points by introducing a slack variable that measures the degree of misclassification. The margin is relaxed to allow for some misclassification, but at the same time, the objective function is modified to minimize the slack variable. To capture this, we use the 'hinge loss function':

$$\max\left(0, 1 - y_i(w^T x_i - b)\right) \tag{10.5}$$

Note that y_i is the ith target (i.e., in this case, 1 or -1), and $w^T x_i - b$ is the ith output.

This function is zero if the constraint in (10.4) is satisfied, in other words, if x_i lies on the correct side of the margin. For data on the wrong side of the margin, the function's value is proportional to the distance from the margin.

The goal of the optimization then is to minimize the following:

$$\lambda\|w\|^2 + \left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(w^T x_i - b)\right)\right] \tag{10.6}$$

where the parameter $\lambda > 0$ determines the trade-off between increasing the margin size and ensuring that the x_i lie on the correct side of the margin.

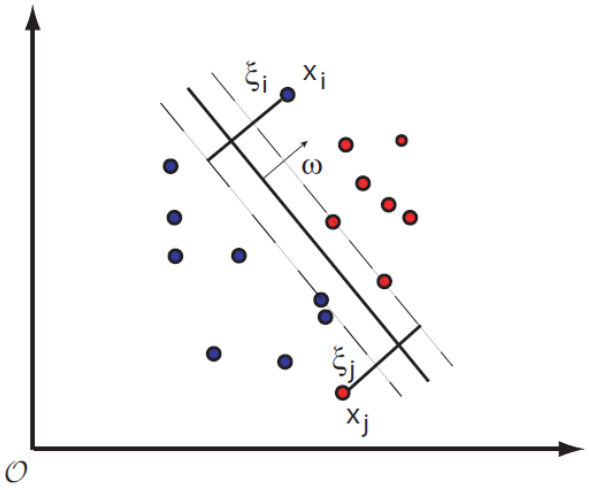
By deconstructing the hinge loss, this optimization problem can be massaged into the following:

The optimal hyperplane is the one that maximizes the margin, subject to the constraint that all data points are correctly classified. The optimization problem that SVM solves can be formulated as:

$$\begin{aligned} &\text{minimize } (\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i) \\ &\text{Subject to:} \\ &y_i(w^T x_i + b) \geq 1 - \xi_i, \forall i = 1, 2, ..., n \\ &\xi_i \geq 0, \forall i = 1, 2, ..., n \end{aligned} \tag{10.7}$$

where C is a hyperparameter that controls the trade-off between maximizing the margin and minimizing the misclassification error, and should be selected by the user,[54] y_i is the class label for the ith data point, x_i is the input vector for the ith data point, and ξ_i is the slack variable that penalizes misclassification.

The objective function $\frac{1}{2}\|w\|^2$ corresponds to the square of the margin, and the term $\sum_{i=1}^{n}\xi_i$ penalizes misclassification. The constraints ensure that all data points are correctly classified, with a margin of at least 1. The slack variable ξ_i allows for some data points to be miss-classified but at a cost to the objective function.[55]



10.3 Lagrangian

As we noticed earlier, the original SVM problem is a constrained optimization problem that seeks to minimize the objective function of the SVM subject to

a set of constraints. In this case, the objective function is the sum of the norm of the weight vector and a regularization term, while the constraints ensure that the decision boundary separates the classes and that the margin is maximized. These constraints are in the form of inequalities, which can make the optimization problem difficult to solve using traditional optimization methods.

The ***Lagrangian*** method can be used to convert the constrained optimization problem into an unconstrained optimization problem. This unconstrained optimization problem can then be solved using a variety of optimization methods, such as gradient descent or Newton's method, to find the optimal solution to the SVM problem. By solving this unconstrained optimization problem, we can obtain the optimal weight vector and bias term that define the decision boundary and maximize the margin between the classes.

The Lagrangian method involves introducing a set of 'Lagrange multipliers' α_i for each constraint, and forming the 'Lagrangian function':

$$L(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i [y_i(w^T x_i + b) - 1] \tag{10.8}$$

where $\alpha = [\alpha_1, \alpha_2, ..., \alpha_n]^T$ is the vector of Lagrange multipliers.

The ***Lagrangian function*** L is a function of the primal variables w and b, as well as the dual variables α. The goal is to find the values of w, b, and α that minimize L, subject to the constraints.

The solution to this optimization problem can be found by taking the partial derivatives of L with respect to w and b, and setting them equal to zero:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0 \tag{10.9}$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{10.10}$$

Solving these equations for w and b in terms of α gives:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{10.11}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{10.12}$$

Substituting these expressions back into the Lagrangian function L yields the dual optimization problem:

$$\text{maximize} \Big(\sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \Big) \tag{10.13}$$

subject to the constraints:

$$0 \le \alpha_i \le C, \forall i = 1, 2, ..., n$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

The solution to this dual optimization problem gives us the values of α that can be used to compute the weight vector w and the bias term b.

The weight of the optimal hyperplane is given by:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i \tag{10.14}$$

and the bias term b can be computed using any support vector x_i that satisfies $0 < \alpha_i < C$:

$$b = y_i - \sum_{j=1}^{n} \alpha_j y_j x_j^T x_i \tag{10.15}$$

The support vectors are the data points that have non-zero Lagrange multipliers. They lie on the margin or on the wrong side of the margin, and they determine the location of the hyperplane. The non-support vectors have zero Lagrange multipliers and do not affect the hyperplane.

10.4 Kernel Functions

The Lagrangian function L introduced above is a special case of a more general type of functions, called ***Kernel functions***, which are schematically defined as follows:

$$\mathcal{L}(\boldsymbol{\alpha}) = \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_{i=1}^{n} \alpha_i \tag{10.16}$$

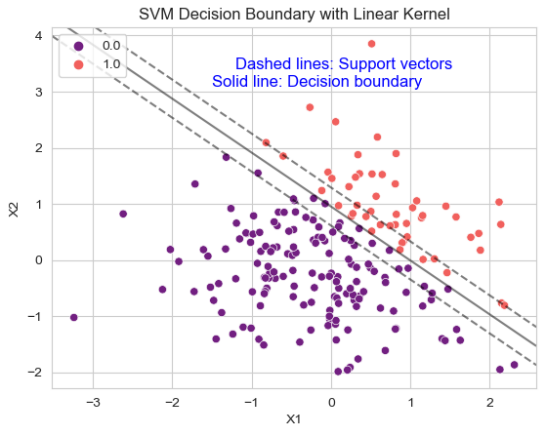
where x_i and x_j are data points, y_i and y_j are their corresponding labels, and K is the kernel function. $\boldsymbol{\alpha}$ is the vector of Lagrange multipliers.

Kernel functions are used in SVMs to transform input data into a higher dimensional space where it can be more easily separated by a hyperplane. The most commonly used kernel functions in SVMs are:

1. *Linear kernel*: This is the simplest kernel function and is used when the data is linearly separable. It has the form:

$$K(x_i, x_j) = x_i^T x_j \tag{10.17}$$

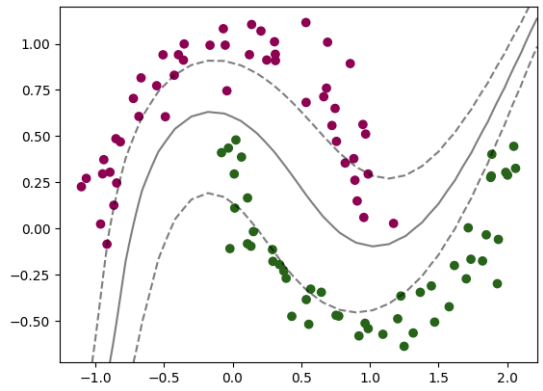
where x_i and x_j are input vectors.



2. *Polynomial kernel*: This kernel function can be used when the data is not linearly separable. It maps the data into a higher dimensional space using a polynomial function. It has the form:

$$K(x_i, x_j) = (a x_i^T x_j + c)^d \tag{10.18}$$

where a, c, and d are hyperparameters that can be tuned to improve the performance of the model.

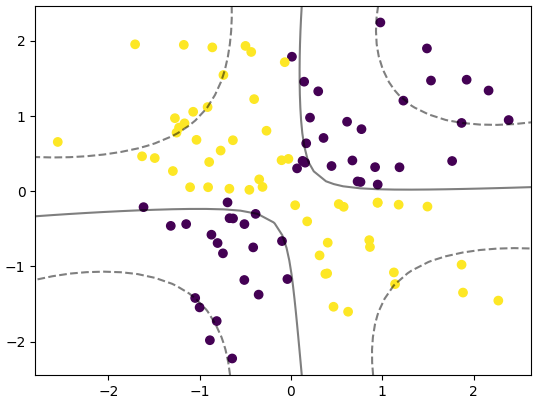


One example of a dataset that is not linearly separable but can be mapped into a higher dimensional space using a polynomial kernel is the "moons" dataset. The moons dataset consists of two interleaving half circles, making it difficult to draw a linear boundary to separate the two classes in two dimensions. However, by mapping the data into a higher dimensional space using a polynomial kernel, it is possible to find a separating hyperplane.

3. *Radial basis function (RBF) kernel*: This is the most commonly used kernel function in SVMs. It can be used when the data is not linearly separable and cannot be mapped into a higher dimensional space using a polynomial kernel. It maps the data into an infinite-dimensional space using a Gaussian function. It has the form.

$$K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2} \tag{10.19}$$

where γ is a hyperparameter that controls the width of the Gaussian function.



One example of a dataset that is not linearly separable and cannot be effectively mapped into a higher dimensional space using a polynomial kernel is the "XOR" dataset. The XOR dataset consists of two classes of points arranged in an "X" shape, such that it is impossible to draw a straight line or a polynomial curve to separate the classes in two dimensions.

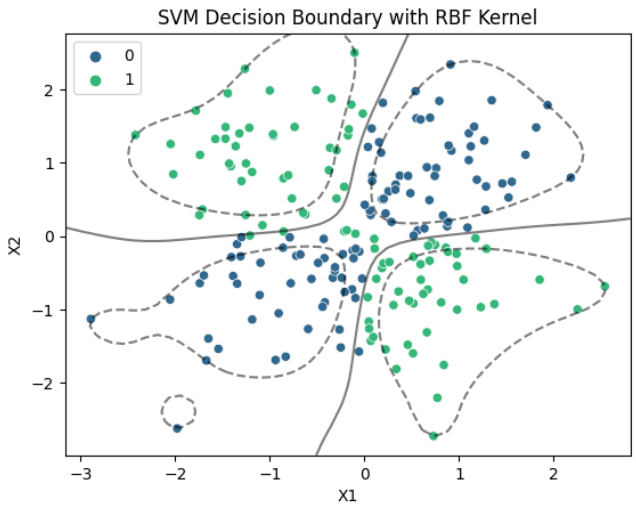
10.5 Python Implementations

Here's a sample Python code to use `SVM` from `sklern`. This code generates a synthetic dataset with two classes, fits an SVM model with an RBF kernel to the data, and plots the data points and decision boundary. Here are the main steps:

```
import numpy as np
import matplotlib.pyplot as plt
```

```
import seaborn as sns
from sklearn.svm import SVC

# Generate synthetic dataset with two classes
X = np.random.randn(200, 2)
y = np.logical_xor(X[:, 0] > 0, X[:, 1] > 0).astype(int)

# Fit SVM model with RBF kernel
model = SVC(kernel='rbf', gamma=1)
model.fit(X, y)

# Plot data points and decision boundary
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=y, palette='viridis')
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx, yy = np.meshgrid(np.linspace(xlim[0], xlim[1], 50), np.linspace(ylim[0], ylim[1], 50))
Z = model.decision_function(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
ax.contour(xx, yy, Z, colors='k', levels=[-1, 0, 1], alpha=0.5,
linestyles=['--', '-', '--'])

# Customize plot
plt.title('SVM Decision Boundary with RBF Kernel')
plt.xlabel('X1')
plt.ylabel('X2')
plt.xlim(xlim)
plt.ylim(ylim)
plt.legend(loc='upper left')

# Show plot
plt.show()
```

This code outputs the following plot:

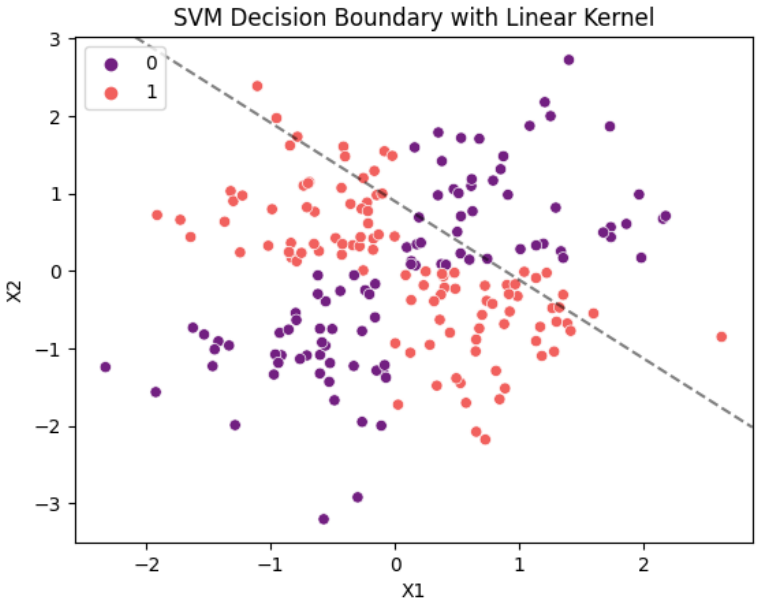


To use a linear or polynomial SVM to run the exact same code, you would need to change the kernel parameter of the SVC class constructor.

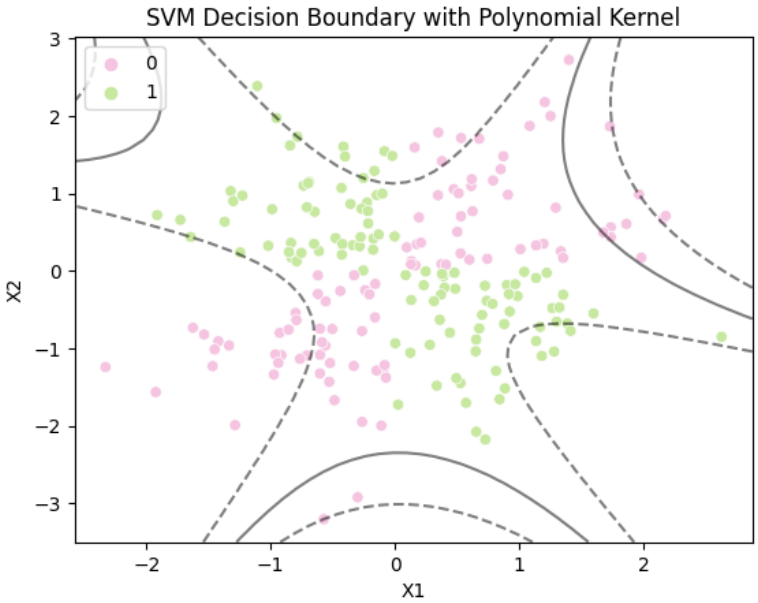
For a linear SVM, you would set the kernel parameter to 'linear', and for a polynomial SVM, you would set the kernel parameter to 'poly'. Additionally, for a polynomial SVM, you may need to adjust the degree parameter to control the degree of the polynomial used in the kernel function.

You would also need to adjust any labels or titles in the plot to reflect the new kernel function being used. Other than these changes, the overall structure of the code would remain the same.

For `kernel='rbf'` (and `palette='magma'`) we get the following:



And for `kernel='poly'` (and `palette='PiYG'`) we get the following plot:



As we can see, the appropriate choice of the kernel has to do with the type of data we're dealing with. Since here we have (intentionally created) XOR data, SVM seems to have best captured it.

Notes

[52]The first two images of this chapter are taken from https://en.wikipedia.org/wiki/Support_vector_machine

[53]If the data is not normalized we could use the following: margin = $\frac{y_i(w^T x_i+b)}{\|w\|}$

[54]C determines the importance given to the two objectives. A smaller value of C will result in a larger margin but more misclassification errors, while a larger value of C will result in a smaller margin but fewer misclassification errors. The value of C needs to be chosen carefully to achieve good performance. If C is set too low, the model may have a large margin but perform poorly on the training set due to misclassification errors. On the other hand, if C is set too high, the model may overfit the training set and generalize poorly to new data. The optimal value of C depends on the specific problem at hand and needs to be determined empirically through techniques such as cross-validation.

[55]The image below is taken from this paper: https://www.worldscientific.com/doi/abs/10.1142/S1469026808002314

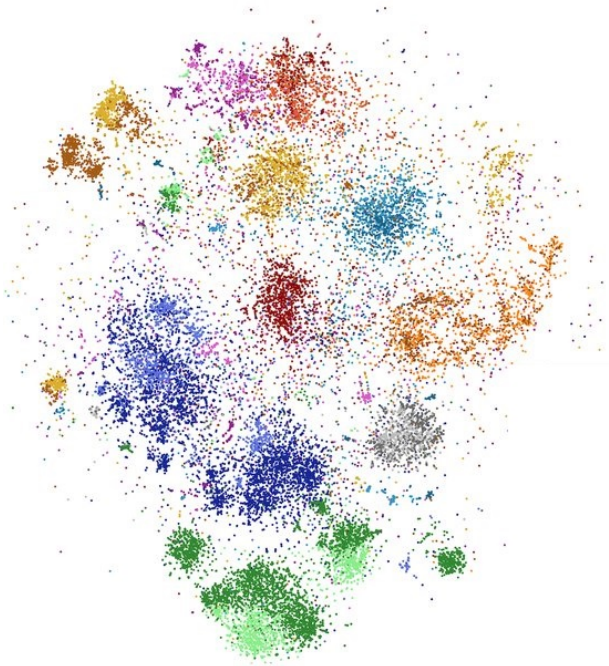
10.6 Further Reading

Here are a few useful resources on Support Vector Machines:

1. *Support Vector Machines Succinctly*, by Alexandre Kowalczyk [20]. This book provides a concise introduction to SVMs, including an overview of the theory behind SVMs and practical examples of SVMs in action.
2. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, by Nello Cristianini and John Shawe-Taylor [10]. This is a classic text on SVMs that covers the fundamentals of SVMs and other kernel-based learning methods. The book is suitable for both beginners and experts in the field.
3. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, by Bernhard Schölkopf and Alexander J. Smola [34]. This book covers the theory behind SVMs and other kernel-based learning methods in detail. The authors also discuss practical applications of these methods and provide a thorough overview of the field.

Chapter 11

Clustering Methods



11.1 Introduction

Clustering is a fundamental unsupervised learning technique in machine learning used to group data points into meaningful subsets or clusters based on their similarity. Clustering methods have a wide range of applications in various domains such as image processing, social network analysis, customer segmentation, and anomaly detection, among others.[56]

Several clustering methods have been developed, each with its own strengths and weaknesses. These clustering methods differ in their approach and underlying assumptions and are suitable for different types of data and applications.

In this chapter, we will introduce some of these methods and their underlying mathematics.

11.2 K-Means Clustering

K-Means Clustering is a widely used unsupervised learning algorithm that partitions a dataset into K clusters based on the mean distance between data points. The goal of K-Means is to minimize the sum of squared distances between each data point and its assigned cluster centroid.

The algorithm works by first initializing K centroids randomly from the data points. Then, in an iterative process, the algorithm assigns each data point to the nearest centroid and recomputes the centroid of each cluster based on the new assignment. This process continues until the centroids no longer move or a maximum number of iterations is reached.

More formally, let $X = x_1, x_2, \ldots, x_n$ be the dataset consisting of n data points in d-dimensional space, and let K be the number of clusters we want to partition the dataset into. The algorithm can be summarized as follows:

1. Initialize K centroids $\mu_1, \mu_2, \ldots, \mu_K$ randomly from the dataset.
2. Assign each data point x_i to the nearest centroid μ_j:

$$c_i = \arg\min_j \|x_i - \mu_j\|^2 \tag{11.1}$$

   where c_i is the index of the cluster that x_i belongs to.[57]
3. Recompute the centroid of each cluster as the mean of the data points assigned to it:

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i \tag{11.2}$$

   where C_j is the set of data points assigned to cluster j.

4. Repeat steps 2 and 3 until the centroids no longer move or a maximum number of iterations is reached.

The objective function of K-Means Clustering is the sum of squared distances between each data point and its assigned cluster centroid, given by:

$$J = \sum_{i=1}^{n} ||x_i - \mu_{c_i}||^2 \tag{11.3}$$

where c_i is the index of the cluster that x_i belongs to, and μ_{c_i} is the centroid of that cluster. The goal of K-Means is to minimize this objective function, so we need to solve the optimization problem:

$$\min_{\mu_1,\ldots,\mu_K} \sum_{i=1}^{n} ||x_i - \mu_{c_i}||^2 \tag{11.4}$$

where c_i is the index of the cluster that x_i belongs to, and μ_{c_i} is the centroid of that cluster.

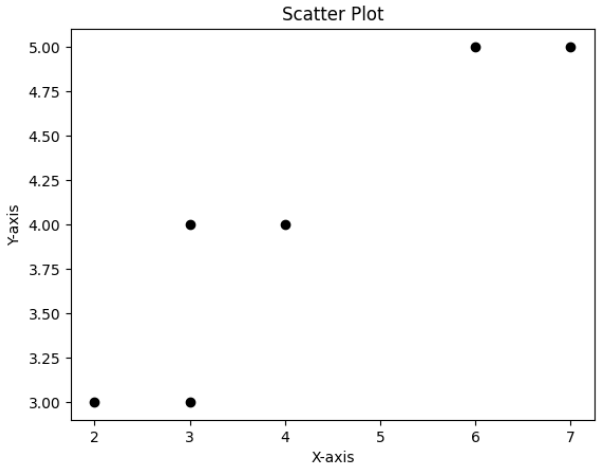
This optimization problem is non-convex and can have multiple local minima. However, the algorithm typically converges to a good local minimum.

One way to initialize the centroids is to randomly select K data points from the dataset as the initial centroids. Another way is to use K-Means++ initialization, which chooses the initial centroids to be far apart from each other. The choice of the number of clusters K is an important hyperparameter in K-Means Clustering. There are several methods to determine the optimal value of K, such as 'the elbow method' and the 'silhouette method'.

K-Means Clustering has several strengths and weaknesses. It is computationally efficient and can scale to large datasets. However, it assumes that clusters are spherical and of equal size, and it is sensitive to the initial centroids.

Example 11.2.1. Let's consider a simple example where we have a dataset with $N = 6$ points in two-dimensional space:

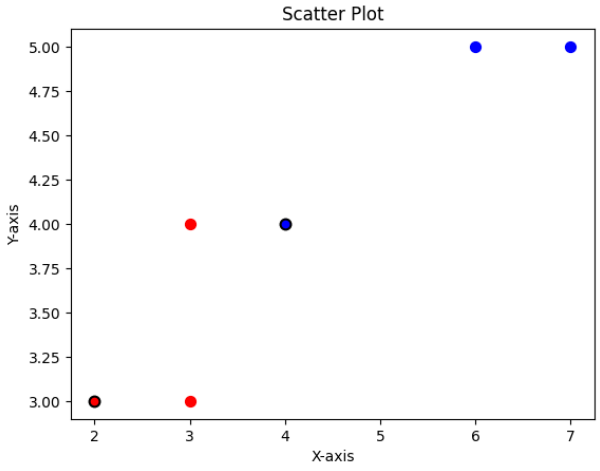
$$\{(2,3),(3,3),(3,4),(4,4),(6,5),(7,5)\}$$



We want to cluster them into $K = 2$ clusters using k-means clustering. We randomly initialize the centroids as $\mu_1 = (2,3)$ and $\mu_2 = (4,4)$, and apply the algorithm.

In the first iteration, we assign each point to its nearest centroid (so, $j \in \{1,2\}$):

$$c_1 = \arg\min_j |x_1 - \mu_j|^2 = 1$$
$$c_2 = \arg\min_j |x_2 - \mu_j|^2 = 1$$
$$c_3 = \arg\min_j |x_3 - \mu_j|^2 = 1$$
$$c_4 = \arg\min_j |x_4 - \mu_j|^2 = 2$$
$$c_5 = \arg\min_j |x_5 - \mu_j|^2 = 2$$
$$c_6 = \arg\min_j |x_6 - \mu_j|^2 = 2$$

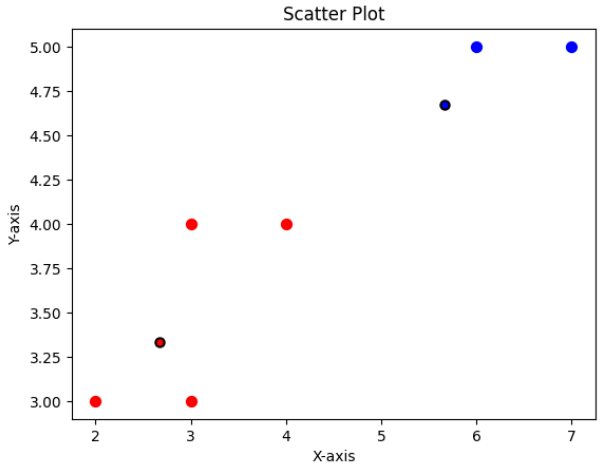


We then update the centroids as the means of the data points assigned to them:

$$\mu_1 = \frac{1}{3}((2,3)+(3,3)+(3,4)) = (2.67, 3.33)$$
$$\mu_2 = \frac{1}{3}((4,4)+(6,5)+(7,5)) = (5.67, 4.67)$$

In the second iteration, we again assign each point to its nearest centroid:

$$c_1 = \arg\min_j |x_1 - \mu_j|^2 = 1$$
$$c_2 = \arg\min_j |x_2 - \mu_j|^2 = 1$$
$$c_3 = \arg\min_j |x_3 - \mu_j|^2 = 1$$
$$c_4 = \arg\min_j |x_4 - \mu_j|^2 = 2$$
$$c_5 = \arg\min_j |x_5 - \mu_j|^2 = 2$$



Based on the new assignments, we compute the new centroids:

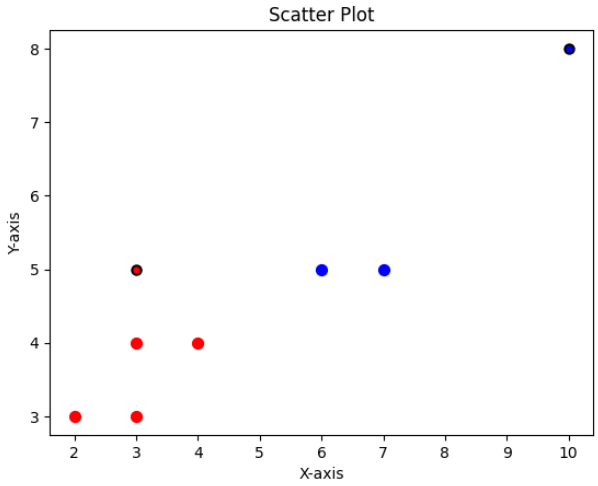
$$\mu_1 = \frac{1}{3}(x_1 + x_2 + x_3) = (3, 5)$$
$$\mu_2 = \frac{1}{2}(x_4 + x_5) = (10, 8)$$

We then repeat the process, assigning each point to its nearest centroid and updating the centroids, until convergence is achieved.

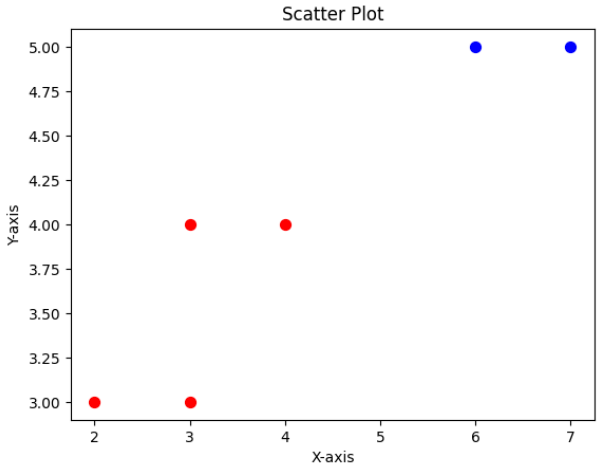
Once the algorithm has converged, we can plot the results to visualize the clusters. In this example, the final cluster assignments and centroids are:

$$c_1 = c_2 = c_3 = 1$$
$$c_4 = c_5 = 2$$
$$\mu_1 = (3, 5)$$
$$\mu_2 = (10, 8)$$

We can plot the original data points along with the final cluster assignments and centroids using a scatter plot, as shown below (see Python codes for K-means in the last section):



In this plot, the blue dots represent the points assigned to cluster 1, the orange dots represent the points assigned to cluster 2, and the black squares represent the centroids. We can see that the k-means algorithm has successfully separated the data points into two distinct clusters.

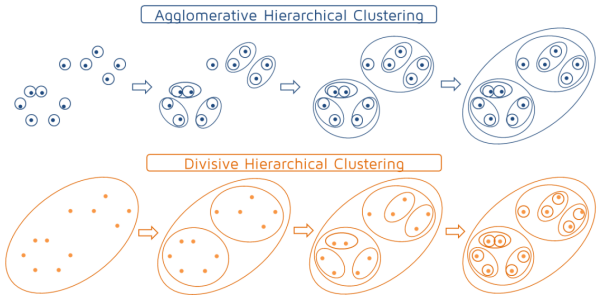


11.3 Hierarchical Clustering

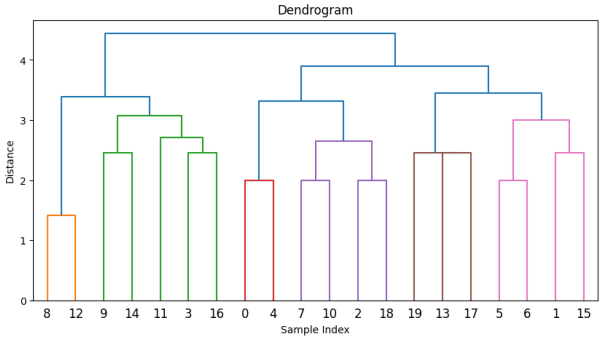
Hierarchical Clustering is a clustering method that seeks to build a hierarchy of nested clusters. It groups similar data points together into smaller and smaller clusters, ultimately leading to a single cluster that contains all the data points.

Hierarchical clustering can be either *agglomerative* (bottom-up) or *divisive* (top-down), with agglomerative clustering being the more common approach. In agglomerative hierarchical clustering, each data point starts out as its own cluster, and

clusters are successively merged together based on a similarity measure until a single cluster containing all data points is formed. In divisive hierarchical clustering, the opposite approach is taken, and a single cluster is repeatedly split into smaller clusters until each data point is in its own cluster.[58]



In hierarchical clustering, the number of clusters is determined by the dendrogram, which is a tree-like diagram that shows the hierarchy of clusters and the order in which they were merged. The dendrogram can be cut at any level to obtain the desired number of clusters.



In what follows, we will focus on the agglomerative approach and its mathematics.

Agglomerative hierarchical clustering algorithm starts by treating each data point as its own cluster, and then iteratively combines the most similar clusters until all data points are part of a single cluster.

The 'similarity' between clusters is usually measured by a distance metric, such as Euclidean distance or cosine similarity. One common way to measure the similarity between two clusters is the 'linkage criterion', which determines the distance between two clusters as a function of the distances between their individual data points.

There are three common types of linkage criteria: single linkage, complete linkage, and average linkage. ***Single linkage*** measures the distance between the closest points in the two clusters, ***complete linkage*** measures the distance between the farthest points in the two clusters, and ***average linkage*** measures the average distance between all pairs of points in the two clusters.

The algorithm continues until all data points are part of a single cluster. At each step, the dendrogram is updated to show the new relationships between the clusters. The dendrogram can be used to determine the optimal number of clusters, by looking for the largest vertical distance between two branches that can be cut while still maintaining a certain level of similarity between clusters.

Mathematically, the agglomerative hierarchical clustering algorithm can be represented as follows:

1. Start with n data points, each as a separate cluster.
2. Compute the pairwise distance matrix between all clusters.
3. Find the two closest clusters based on the chosen linkage criterion.
4. Merge the two closest clusters into a single cluster.
5. Recompute the pairwise distance matrix between the new cluster and all other clusters.
6. Repeat steps 3-5 until all data points are part of a single cluster.

The linkage criteria can be represented mathematically as follows:

1. *Single linkage:*

$$d_{SL}(C_i, C_j) = \min_{x \in C_i, y \in C_j} d(x, y) \tag{11.5}$$

2. *Complete linkage:*

$$d_{CL}(C_i, C_j) = \max_{x \in C_i, y \in C_j} d(x, y) \tag{11.6}$$

3. *Average linkage:*

$$d_{AL}(C_i, C_j) = \frac{1}{|C_i| \cdot |C_j|} \sum_{x \in C_i, y \in C_j} d(x, y) \tag{11.7}$$

where C_i and C_j are two clusters, $d(x, y)$ is the distance between two data points x and y, and $|C_i|$ and $|C_j|$ are the number of data points in the clusters C_i and C_j, respectively.

We close this section with a few remarks on the comparison between K-means and hierarchical clustering. The main difference between hierarchical clustering and K-means clustering is that the former does not require a pre-specified number of clusters, whereas the latter requires the user to specify a *fixed* number of clusters.

Hierarchical clustering also has the advantage of being able to handle non-linearly separable data and provide a visual representation of the hierarchical relationships between clusters. However, it can be computationally expensive for large datasets and the resulting dendrograms can be difficult to interpret. K-means clustering, on the other hand, is computationally efficient and can handle large datasets, but may not work well on datasets with irregular shapes or clusters that are not well-separated.

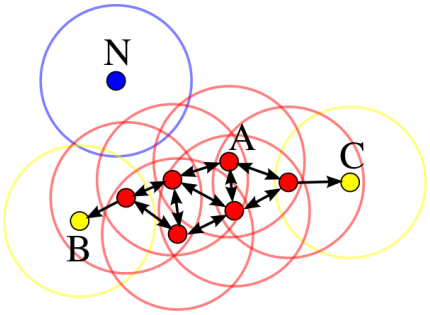
11.4 DBSCAN

Density-Based Spatial Clustering of Applications with Noise (***DBSCAN***) is a clustering algorithm that groups together data points that are close to each other in a high-density region, while marking data points that lie alone in low-density regions as noise. DBSCAN is particularly useful for discovering clusters of arbitrary shape, handling noise, and working well with large datasets.

The DBSCAN algorithm works by defining a neighborhood around each point and identifying dense regions that have a minimum number of points within that neighborhood. These dense regions are known as *core points*, and the neighboring points that are within the defined neighborhood are known as *border points*. Points that are not within the neighborhood of any core or border point are considered noise.

The algorithm requires two main input parameters: the radius of the neighborhood (epsilon) and the minimum number of points required to form a dense region (minPts). DBSCAN proceeds as follows:

1. For each point in the dataset, calculate the distance between that point and all other points in the dataset.
2. For each point, identify all other points within a radius epsilon of that point. This creates a neighborhood around each point.
3. If the number of points in the neighborhood of a given point is greater than or equal to the minimum number of points required to form a dense region (minPts), then that point is considered a core point.
4. For each core point, all points within its neighborhood (including other core points) are assigned to the same cluster.
5. Points that are not core points but lie within the neighborhood of a core point are considered border points and are also assigned to the same cluster as the core point.
6. Points that are not core points and do not lie within the neighborhood of any core point are marked as noise and not included in any cluster.



In the diagram above we have minPts = 4.[59] Point A and the other red points are our core points because the area surrounding these points in an ϵ radius contains at least 4 points (including the point itself). Because they are all reachable from one another, they form a single cluster. Points B and C are not core points, but are reachable from A (via other core points) and thus belong to the cluster as well. Point N is a noise point that is neither a core point nor directly reachable.

The DBSCAN algorithm can be mathematically represented as follows:

Let D be the dataset, epsilon be the radius of the neighborhood, and minPts be the minimum number of points required to form a dense region. Let $N(p)$ be the neighborhood of point p, defined as all points within distance epsilon of p, and let $|N(p)|$ be the cardinality of $N(p)$. Then, the DBSCAN algorithm can be defined as follows:

1. Define a set of unvisited points $U = D$.
2. Initialize an empty set of clusters C.
3. While U is not empty:
    a. Select a random point p from U and mark it as visited.
    b. If $|N(p)| <$ minPts, mark p as noise.
    c. Otherwise, create a new cluster C' and add p to C'.
    d. For each point q in $N(p)$:
        d1. If q is unvisited, mark q as visited.
        d2. If $|N(q)| >=$ minPts, add q to C'.
        d3. If q is not already a member of any cluster, add q to C'.
    e. Remove all points in C' from U.

4. Output the set of clusters C.

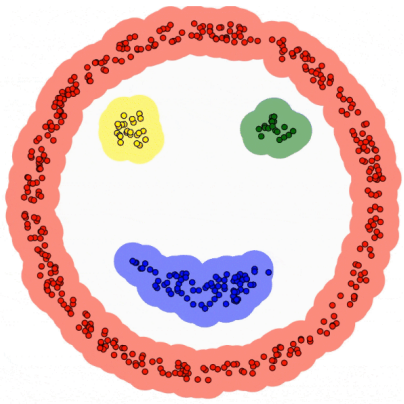
DBSCAN has several advantages over other clustering algorithms like K-means or hierarchical clustering. Firstly, it can handle clusters of arbitrary shapes and sizes, unlike K-means which assumes clusters are spherical and of equal size. Additionally, DBSCAN does not require the number of clusters to be specified beforehand, unlike K-means.

Furthermore, DBSCAN is capable of handling noise and outliers, as it labels data points that do not belong to any cluster as noise. This is in contrast to K-means and hierarchical clustering, which assign all data points to clusters, including outliers.

Another advantage of DBSCAN is its ability to detect clusters with varying densities. This is accomplished by defining a neighborhood radius, which determines the size of the clusters. Points that are within this radius of a core point are classified as part of the same cluster, allowing the algorithm to detect clusters with different densities.

When dealing with linearly non-separable data, such as datasets with complex shapes or overlapping clusters, DBSCAN is generally considered to be a better choice than hierarchical clustering. This is because DBSCAN is designed to identify clusters based on the local density of data points, rather than their global distances or similarities, which makes it more robust to non-linear shapes and overlapping clusters.[60]

Here's an example of a linearly non-separable data that is well-clustered using DBSCANL[61]



11.5 Python Implementations

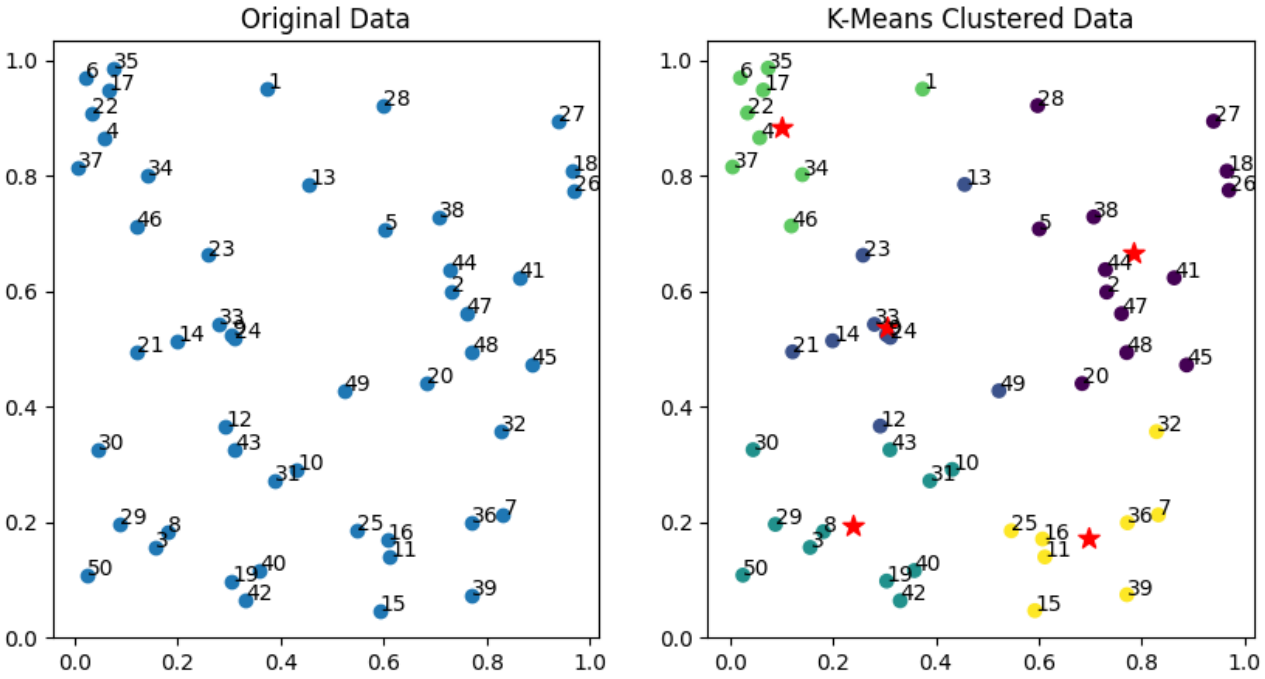
11.5.1 K-Means clustering

Here's an example of the K-means algorithm in Python, with the number of clusters set to 5:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Generate some random data
np.random.seed(42)
X = np.random.rand(50, 2)

# Define the number of clusters
k = 5

# Create a k-means model and fit it to the data
kmeans = KMeans(n_clusters=k, random_state=42).fit(X)

# Get the cluster assignments
clusters = kmeans.labels_

# Plot the original data with labels
plt.figure(figsize=(10, 5))
plt.subplot(121)
for i in range(X.shape[0]):
```

```
    plt.text(X[i, 0], X[i, 1], str(i+1), color='black', fontsize=10)
plt.scatter(X[:,0], X[:,1])
plt.title('Original Data')

# Get the cluster centroids
centroids = kmeans.cluster_centers_

# Plot the clustered data with different colors for each cluster and labels
plt.subplot(122)
for i in range(X.shape[0]):
    plt.text(X[i, 0], X[i, 1], str(i+1), color='black', fontsize=10)
plt.scatter(X[:,0], X[:,1], c=clusters)

# Plot the centroids with red markers
plt.scatter(centroids[:,0], centroids[:,1], marker='*', c='r', s=100)
plt.title('K-Means Clustered Data')
plt.show()

# Check the cluster assignments of each data point
#for i, c in enumerate(clusters):
    # print(f"Data point {i+1} assigned to cluster {c}")
```

The resulting plot should show the four clusters and their respective centroids. Here's the output of this code:

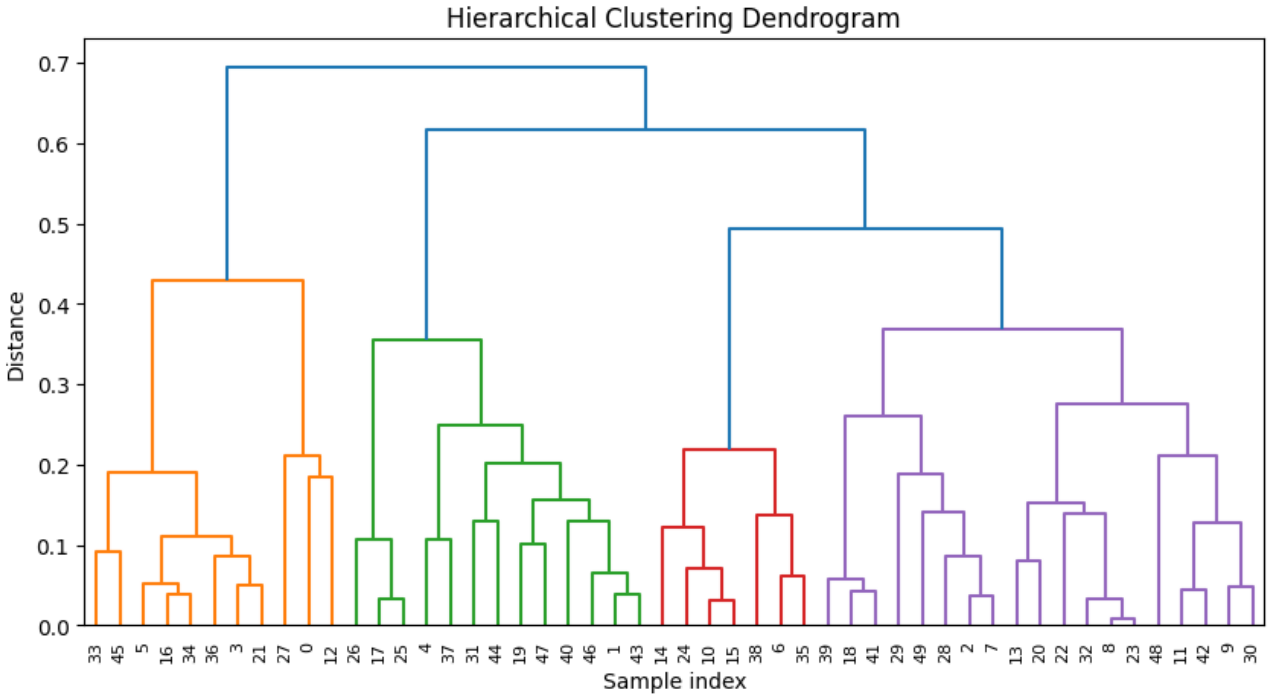


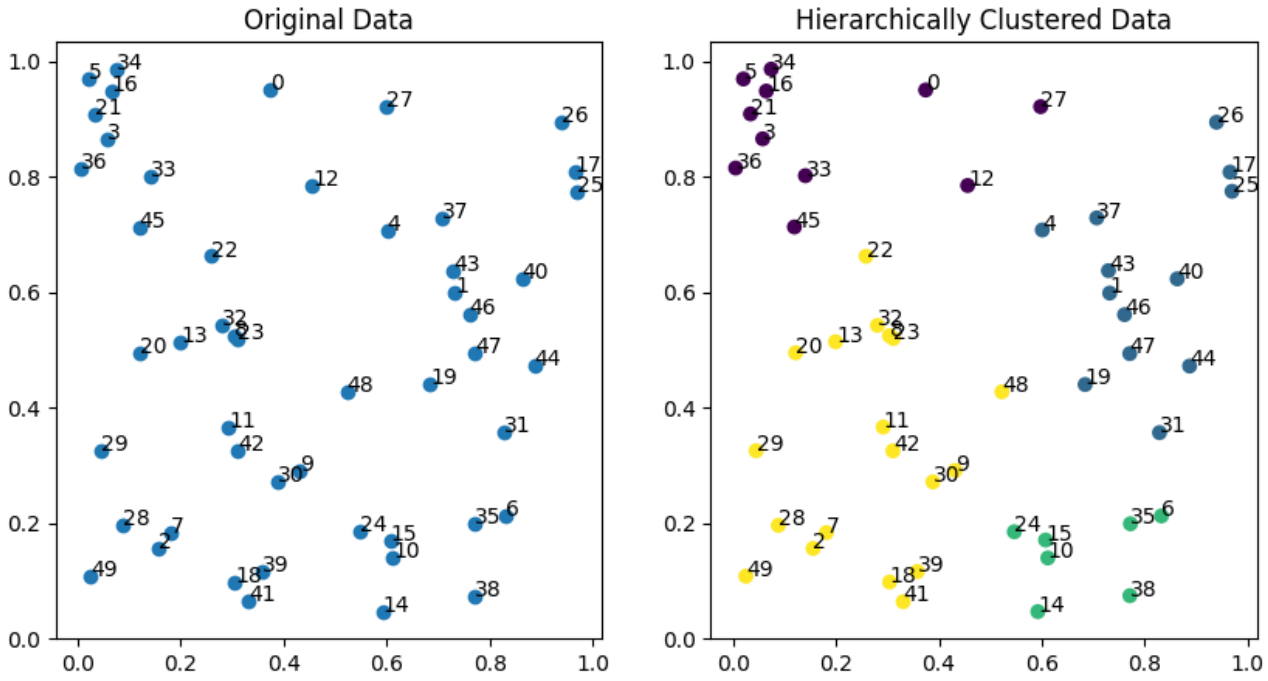
11.5.2 Hierarchical Clustering

Here's an example of hierarchical clustering implementation in Python, using the 'average' method (other methods can similarly be called directly by their names in the specification of the `method` argument in `linkage`). I have also separately clustered the data using scatter plots in exactly the same way that the dendrogram has performed its clustering:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

# Generate some random data
np.random.seed(42)
X = np.random.rand(50, 2)

# Calculate the linkage matrix using the ward method
Z = linkage(X, method='average')

# Plot the dendrogram to determine the optimal number of clusters
plt.figure(figsize=(10, 5))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Sample index')
plt.ylabel('Distance')
dendrogram(Z)
plt.show()

# Get the cluster assignments
k = 4  # specify the number of clusters
clusters = fcluster(Z, k, criterion='maxclust')

# Plot the original data with labels
plt.figure(figsize=(10, 5))
plt.subplot(121)
for i in range(X.shape[0]):
    plt.text(X[i, 0], X[i, 1], str(i), color='black', fontsize=10)
plt.scatter(X[:,0], X[:,1])
plt.title('Original Data')

# Plot the clustered data with different colors for each cluster and labels
plt.subplot(122)
for i in range(X.shape[0]):
    plt.text(X[i, 0], X[i, 1], str(i), color='black', fontsize=10)
plt.scatter(X[:,0], X[:,1], c=clusters)
plt.title('Hierarchically Clustered Data')

plt.show()

# Check the cluster assignments of each data point
# for i, c in enumerate(clusters):
   # print(f"Data point {i+1} assigned to cluster {c}")
```

Here's the output of the code:





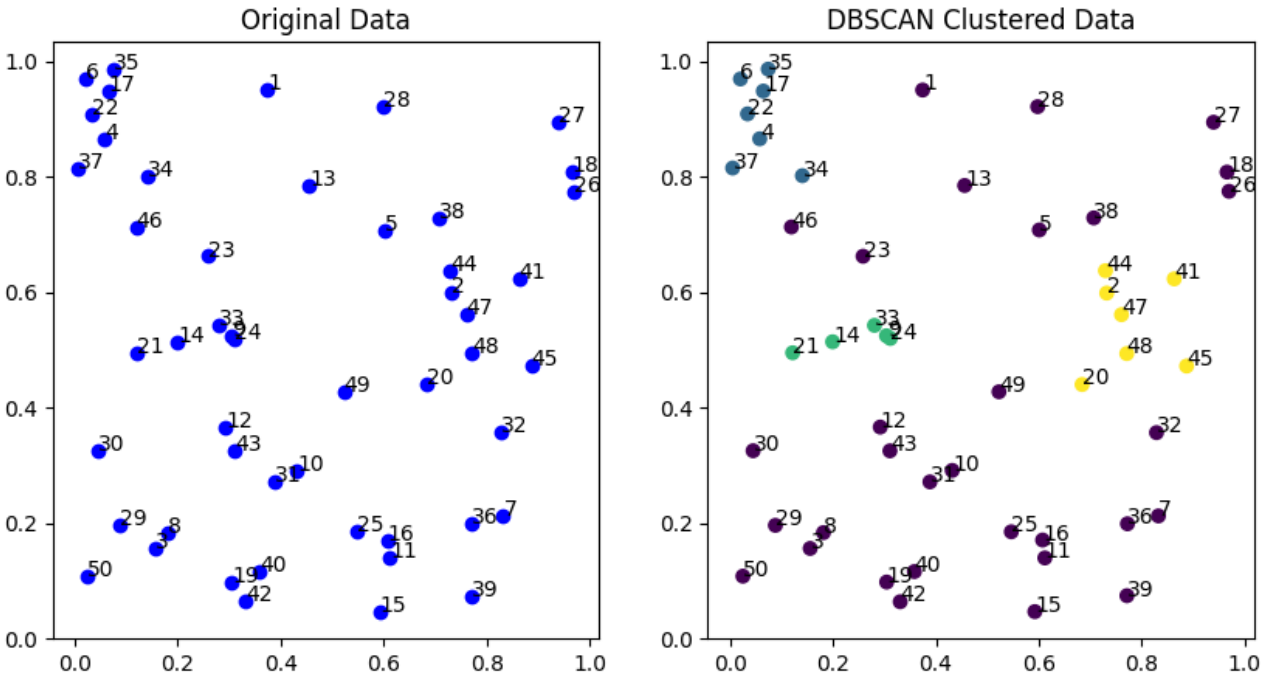
11.5.3 DBSCAN Clustering

Here's the same data set that has been classified with DBSCAN, with the essential parameters set to `eps=0.1` and `min_samples=5` (you can play with these parameters to see how the resulting clusters change):

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.cluster import DBSCAN

# Generate some random data
np.random.seed(42)
X = np.random.rand(50, 2)

# Instantiate and fit the DBSCAN model
db = DBSCAN(eps=0.12, min_samples=5)
db.fit(X)

# Get the cluster assignments
clusters = db.labels_

# Plot the original data with labels
plt.figure(figsize=(10, 5))
plt.subplot(121)
for i in range(X.shape[0]):
    plt.text(X[i, 0], X[i, 1], str(i+1), color='black', fontsize=10)
plt.scatter(X[:,0], X[:,1], c='blue')
plt.title('Original Data')

# Plot the clustered data with different colors for each cluster and labels
plt.subplot(122)
for i in range(X.shape[0]):
    plt.text(X[i, 0], X[i, 1], str(i+1), color='black', fontsize=10)
plt.scatter(X[:,0], X[:,1], c=clusters)
plt.title('DBSCAN Clustered Data')

plt.show()
```

Here's the output:



Notes

[56]The image above is taken from this research paper: https://aclanthology.org/2020.acl-main.493/

[57]Remember, the argmin function returns the argument (input value) that minimizes a given function.

[58]The image below is taken from https://quantdare.com/hierarchical-clustering/

[59]This is taken from https://en.wikipedia.org/wiki/DBSCAN

[60]For this reason, DBSCAN is also computationally efficient, particularly for large datasets, as it only needs to calculate pairwise distances for points within a certain radius. This reduces the number of distance calculations required compared to K-means and hierarchical clustering, which calculate distances between all pairs of data points.

[61]the image is taken from here: https://ml-explained.com/blog/dbscan-explained

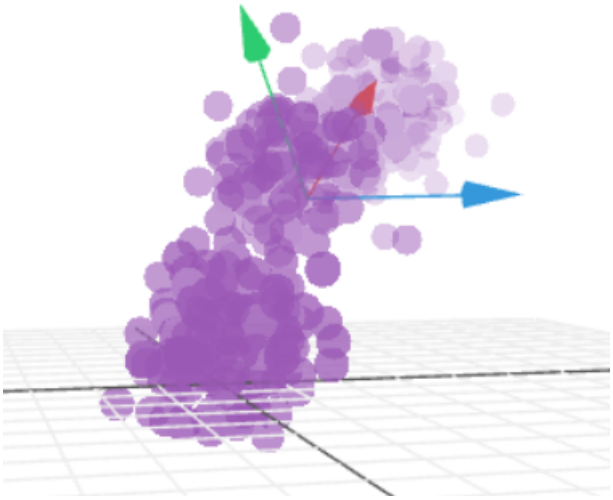
11.6 Further Reading

Here are a few references on clustering methods, particularly on K-Means Clustering, Hierarchical Clustering and Density-Based Spatial Clustering of Applications with Noise (DBSCAN):

1. *Pattern Recognition and Machine Learning*, by Christopher Bishop [5]. This textbook provides an in-depth treatment of the mathematical foundations of various clustering algorithms, including K-Means Clustering, Hierarchical Clustering and DBSCAN. It includes detailed discussions on the underlying probability distributions, optimization algorithms and other technical details.
2. *Data Mining: Concepts and Techniques*, by Jiawei Han, Micheline Kamber, and Jian Pei [14]. This book covers a wide range of data mining techniques, including clustering algorithms like K-Means, Hierarchical Clustering and DBSCAN. It provides a practical approach to implementing these algorithms, with examples and case studies.
3. *Introduction to Data Mining*, by Pang-Ning Tan, Michael Steinbach, and Vipin Kumar [41]. This textbook provides an introduction to the concepts and techniques of data mining, including clustering algorithms like K-Means, Hierarchical Clustering and DBSCAN. It covers the underlying mathematical foundations, as well as practical issues in implementing these algorithms.
4. "Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications", by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu [33]. This paper introduced the DBSCAN algorithm, which is a density-based clustering algorithm. It provides a detailed mathematical analysis of the algorithm and its properties, and includes experimental results on a variety of datasets. This paper is often cited as one of the seminal works in the field of density-based clustering.

Chapter 12

Principal Component Analysis



12.1 Introduction

Principal Component Analysis (***PCA***) is a statistical technique that is commonly used for reducing the dimensionality of a dataset. It is a way of finding the underlying structure or patterns in the data, and representing them in a more compact form. PCA is based on the idea that the most important information in a dataset can be captured by a small number of linear combinations of the original variables.[62]

PCA involves a mathematical procedure that transforms a set of possibly correlated variables into a set of uncorrelated variables called 'principal components'. The first principal component is the linear combination of the original variables that captures the most variation in the data. The second principal component is the next linear combination that captures the most variation, subject to being orthogonal (i.e., uncorrelated) to the first principal component, and so on for subsequent principal components.

PCA can be used for various purposes, such as data visualization, data compression, and data preprocessing. In data visualization, the first two or three principal components can be plotted to visualize the structure or patterns in high-dimensional data. In data compression, the principal components can be used to represent the data in a more compact form, reducing the amount of storage or memory required. In data preprocessing, PCA can be used to remove redundant or irrelevant variables from the data, making it easier to analyze.

In this chapter, we will discuss the underlying mathematics of PCA and the implementation of the algorithm in Python.

12.2 The Mathematics of PCA

The mathematics behind PCA involves some of the notions we learned from the chapters on linear algebra and statistics earlier in this book. They involve calculating the mean vector, covariance matrix, eigenvectors, and eigenvalues of the data, and projecting the data onto the eigenvectors to obtain the principal components.

The eigenvectors of the covariance matrix are the 'directions' in which the data vary the most, and the eigenvalues represent the amount of variation in the data along each eigenvector. The first principal component is the eigenvector with the largest eigenvalue, and so on for subsequent principal components.

Performing PCA includes the following steps:

1. *Standardizing data.* Standardizing data is a common practice in PCA for two main reasons:

   (a) Variance normalization: PCA is based on the variance-covariance matrix of the data. If the variables are measured on different scales, some variables with

larger variance may dominate the analysis. Standardizing the data (subtracting the mean and dividing by the standard deviation) ensures that all variables are on the same scale and have equal variance, which allows for a fair comparison between them.

(b) Interpretability: When interpreting the principal components, it is important to note that their coefficients (loadings) represent the correlation between each variable and the component. If the variables are measured on different scales, their loadings will not be directly comparable. Standardizing the data ensures that the loadings can be compared directly and their magnitude represents the strength of the relationship between each variable and the component.

The standardization formula used in PCA is:

$$\boldsymbol{X}_{std} = \frac{\boldsymbol{X} - \bar{\boldsymbol{x}}}{\boldsymbol{s}} \tag{12.1}$$

where $\boldsymbol{X}$ is the original data matrix, $\bar{\boldsymbol{x}}$ is the vector of column means, and $\boldsymbol{s}$ is the vector of column standard deviations. This formula scales each variable to have mean 0 and standard deviation 1.

2. *Calculation of the covariance matrix.* We use the formula

$$\boldsymbol{\Sigma} = \frac{1}{n-1}\boldsymbol{X}_{std}^T\boldsymbol{X}_{std} \tag{12.2}$$

to calculate the covariance matrix in PCA. The covariance matrix formula computes the covariance between each pair of variables in the dataset.[63]

3. *Calculation of the eigenvectors and eigenvalues of the covariance matrix.* We do this using the following formula:

$$\boldsymbol{\Sigma}\boldsymbol{v} = \lambda\boldsymbol{v} \tag{12.3}$$

where λ is an eigenvalue of the covariance matrix $\boldsymbol{\Sigma}$, and $\boldsymbol{v}$ is the corresponding eigenvector. The eigenvectors of the covariance matrix represent the directions of maximum variation in the data, while the eigenvalues represent the amount of variation along each direction.

4. *Calculation of the principal components.* We use the following formula to this end:

$$\boldsymbol{z_i} = \boldsymbol{v}_i^T(\boldsymbol{x_i} - \bar{\boldsymbol{x}}) \tag{12.4}$$

where $\boldsymbol{z_i}$ is the ith principal component of the dataset, and $\boldsymbol{v}_i$ is the eigenvector corresponding to the ith largest eigenvalue. The principal component formula calculates the projections of each data point onto the eigenvectors of the covariance matrix. The projections are the principal components of the data, which represent the data in a new coordinate system that captures the maximum variation.

Example 12.2.1. Suppose we have a four-dimensional dataset with variables—X_1 (height, in ft), X_2 (weight, in lbs), X_3 (age, in years), and X_4 (income, in USD)—for 5 individuals:

$$\boldsymbol{X} = \begin{bmatrix} X_1 & X_2 & X_3 & X_4 \\ 6 & 150 & 25 & 50000 \\ 5.5 & 120 & 30 & 45000 \\ 6.1 & 180 & 35 & 60000 \\ 5.8 & 160 & 27 & 55000 \\ 6.2 & 200 & 40 & 70000 \end{bmatrix}$$

Suppose we want to reduce this data into two principal components.

First, we need to standardize the data by centering it at the mean and scaling it by the standard deviation of each variable:

$$\boldsymbol{X}_{std} = \frac{\boldsymbol{X} - \bar{\boldsymbol{x}}}{\boldsymbol{s}}$$

where $\bar{\boldsymbol{x}}$ is the mean vector of the columns and $\boldsymbol{s}$ is the vector of standard deviations of the columns:

$$\bar{\boldsymbol{x}} = \begin{bmatrix} 5.92 & 162 & 31.4 & 54000 \end{bmatrix}$$

$$\boldsymbol{s} = \begin{bmatrix} 0.26726124 & 29.69848481 & 6.42910104 & 9984.77919594 \end{bmatrix}$$

Substituting the values, we get:

$$\boldsymbol{X}_{std} = \begin{bmatrix} 0.281818 & -0.333837 & -0.705658 & -0.178317 \\ -1.254545 & -1.00168 & -0.117444 & -0.357709 \\ 0.763636 & 0.667674 & 0.981076 & 0.535631 \\ -0.236363 & -0.222019 & -0.529277 & 0.178317 \\ 1.045454 & 0.889864 & 0.3713021 & 0.82207 \end{bmatrix}$$

Next, we calculate the covariance matrix:

$$\boldsymbol{\Sigma} = \frac{1}{n-1}\boldsymbol{X}_{std}^T\boldsymbol{X}_{std}$$

Substituting the values, we get:

$$\boldsymbol{\Sigma} = \begin{bmatrix} 1.000000 & 0.965225 & -0.957090 & 0.925617 \\ 0.965225 & 1.000000 & -0.993936 & 0.970859 \\ -0.957090 & -0.993936 & 1.000000 & -0.970638 \\ 0.925617 & 0.970859 & -0.970638 & 1.000000 \end{bmatrix}$$

We then find the eigenvalues and eigenvectors of the covariance matrix:

$$\det(\boldsymbol{\Sigma} - \lambda\boldsymbol{I}) = 0$$

where $\boldsymbol{I}$ is the 4×4 identity matrix. Solving this equation gives us the eigenvalues:

$$\lambda_1 = 3.8109, \lambda_2 = 0.1181, \lambda_3 = 0.0607, \lambda_4 = 0.0103$$

When we perform PCA, we typically sort the eigenvalues in decreasing order to identify the most important principal components. The eigenvalues represent the amount of variance explained by each principal component, and the larger the eigenvalue, the more important the corresponding principal component is in explaining the variability in the data.

In our case, since we want to reduce a four-dimensional dataset to two principal components, so we need to select the two largest eigenvalues, which correspond to the two most important principal components. In this case, the two largest eigenvalues are $\lambda_1 = 3.8109$ and $\lambda_2 = 0.1181$, so we should work with these eigenvalues.

The first principal component will account for

$$\frac{3.8109}{(3.8109 + 0.1181 + 0.0607 + 0.0103)} = \frac{3.8109}{4} = 95.1\%$$

of the total variance, and the second principal component will account for

$$\frac{0.1181}{3.8109 + 0.1181 + 0.0607 + 0.0103)} = \frac{0.1181}{4} = 2.9\%$$

of the total variance.

So, with λ_1 and λ_2 alone, we can cover a total of 98% of the total variance of the data. The remaining two eigenvalues are relatively small, so it seems like choosing 2 as the number of principal components was a good choice.

To compute the eigenvectors corresponding to the eigenvalues $\lambda_1 = 3.8109$ and $\lambda_2 = 0.1181$ using the equation

$$(\boldsymbol{\Sigma} - \lambda_i \boldsymbol{I})\boldsymbol{v}_i = \boldsymbol{0}$$

where $\boldsymbol{I}$ is the identity matrix and $i = 1, 2$, we need to solve the following two equations:

For $\lambda_1 = 3.8109$, we have:

$$\begin{bmatrix} -2.8109 & 0.965225 & -0.957090 & 0.925617 \\ 0.965225 & -2.8109 & -0.993936 & 0.970859 \\ -0.957090 & -0.993936 & -2.8109 & -0.970638 \\ 0.925617 & 0.970859 & -0.970638 & -2.8109 \end{bmatrix} \begin{bmatrix} x \; y \; z \; w \end{bmatrix}$$

$$= \begin{bmatrix} 0 \; 0 \; 0 \; 0 \end{bmatrix}$$

For $\lambda_2 = 0.1181$, we have:

$$\begin{bmatrix} 0.8819 & 0.965225 & -0.957090 & 0.925617 \\ 0.965225 & 0.8819 & -0.993936 & 0.970859 \\ -0.957090 & -0.993936 & 0.8819 & -0.970638 \\ 0.925617 & 0.970859 & -0.970638 & 0.8819 \end{bmatrix} \begin{bmatrix} x \; y \; z \; w \end{bmatrix}$$

$$= \begin{bmatrix} 0 \; 0 \; 0 \; 0 \end{bmatrix}$$

After solving these systems of equations, we obtain the eigenvectors:

$$\boldsymbol{v}_1 = [0.5000, 0.5223, -0.4989, 0.4771]^T$$
$$\boldsymbol{v}_2 = [-0.3557, -0.3381, -0.3646, -0.8138]^T$$

Therefore, the eigenvectors corresponding to λ_1 and λ_2 are:

$$\boldsymbol{v}_1 = \begin{bmatrix} 0.5000 \\ 0.5223 \\ -0.4989 \\ 0.4771 \end{bmatrix}, \; \boldsymbol{v}_2 = \begin{bmatrix} -0.3557 \\ -0.3381 \\ -0.3646 \\ -0.8138 \end{bmatrix}$$

The eigenvectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ obtained in the previous step represent the directions along which the data varies the most, in decreasing order of variability. Specifically, $\boldsymbol{v}_1$ represents the direction of maximum variance, and $\boldsymbol{v}_2$ represents the direction of the second maximum variance, orthogonal to $\boldsymbol{v}_1$.

In the context of the original dataset, the variables can be thought of as different dimensions, and the individuals can be thought of as points in this multi-dimensional space. The eigenvector $\boldsymbol{v}_1$ can be used to project the data onto a one-dimensional subspace that captures most of the variability in the data, and $\boldsymbol{v}_2$ can be used to project the data onto a two-dimensional subspace that captures the next most variability.

To find the combinations of the original variables $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, we can calculate the dot product of each eigenvector with the mean-centered dataset. Using the formula $\boldsymbol{z_i} = \boldsymbol{v}_i^T(\boldsymbol{x_i} - \bar{\boldsymbol{x}})$, we get:

$$\boldsymbol{z}_1 = \begin{bmatrix} 0.5177 \\ 0.4992 \\ -0.4411 \\ 0.5378 \end{bmatrix}, \boldsymbol{z}_2 = \begin{bmatrix} -0.8161 \\ 0.2564 \\ -0.2609 \\ 0.4653 \end{bmatrix}$$

Let's recall the original variables X_1 (height), X_2 (weight), X_3 (age), and X_4 (income). Then the linear combinations of these variables that correspond to $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are:

$$\text{PC}_1 = 0.5177X_1 + 0.4992X_2 - 0.4411X_3 + 0.5378X_4$$
$$\text{PC}_2 = -0.816X_1 + 0.2564X_2 - 0.2609X_3 + 0.4653X_4$$

Note that the coefficients of the linear combinations represent the 'loadings' of each variable on each principal component. These loadings indicate the contribution of each variable to each component and can be used to interpret the meaning of the components in the context of the original variables.

12.3 Final Remarks

We conclude this chapter with a few remarks on PCA and its limitations. One limitation of PCA is that it assumes that the underlying data has a linear structure. If the data has a nonlinear structure, PCA may not be the best approach. In such cases, other techniques such as nonlinear dimensionality reduction may be more appropriate.

Another limitation of PCA is that the interpretation of the principal components can be difficult. The principal components are linear combinations of the original variables, and their meaning may not be immediately obvious. However, the interpretation of the principal components can be aided by examining the loadings of the variables on each principal component. The loadings indicate the extent to which each variable contributes to each principal component.

In summary, PCA is a powerful technique for reducing the dimensionality of a dataset while retaining the most important information. It is based on finding the eigenvectors and eigenvalues of the covariance matrix of the data, and the resulting principal components are uncorrelated linear combinations of the original variables. PCA can be used for various purposes, such as data visualization, data compression, and data preprocessing. However, it has limitations such as assuming a linear structure and the difficulty of interpreting the principal components.

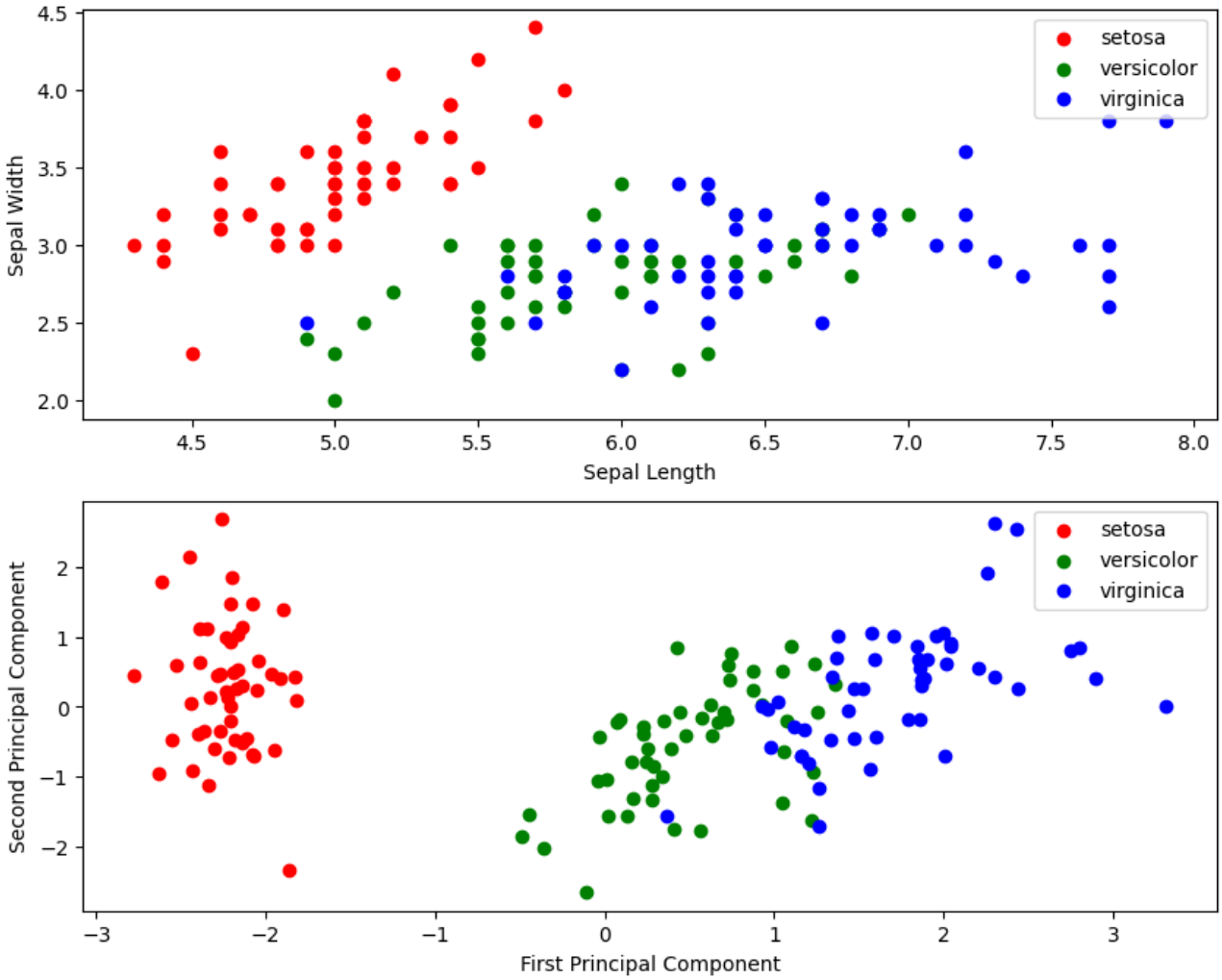
12.4 Python Implementations

Here's an example implementation of PCA on the iris dataset:

```python
# import required libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler


# load the iris dataset
iris = load_iris()

# separate the features and labels
X = iris.data
y = iris.target

# X is a four-dimensional data, of the following form:

# array([[5.1, 3.5, 1.4, 0.2],
#        [4.9, 3. , 1.4, 0.2],
#        [4.7, 3.2, 1.3, 0.2],
#        [4.6, 3.1, 1.5, 0.2],
#        [5. , 3.6, 1.4, 0.2],
#        [5.4, 3.9, 1.7, 0.4],
#        [4.6, 3.4, 1.4, 0.3],
#        [5. , 3.4, 1.5, 0.2],
#        ..................]])

# where the first_column is sepal length (cm), then second is sepal width (cm),
# the third is petal length (cm), and the last one is petal width (cm).
# you can get these names by running this code: iris.feature_names
```

```
# y represents the codes for the target flowers:

# array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
#        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
#        0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
#        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
#        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
#        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
#        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

# where 0 represents 'setosa', 1 represents 'versicolor' and 2 represenrts
# 'virginica'.


figure, axes = plt.subplots(2, figsize=(10, 8))

# plot the original data, with legends
targets = iris.target_names
colors = ['r', 'g', 'b']
for target, color in zip(targets, colors):
    indices = np.where(y == np.where(iris.target_names == target)[0][0])
    axes[0].scatter(X[indices, 0], X[indices, 1], c=color, label=target, alpha=1)
axes[0].legend(handles=handles, loc='best')
axes[0].set_xlabel('Sepal Length')
axes[0].set_ylabel('Sepal Width')

# create a scaler object and fit it to the data
scaler = StandardScaler()
scaler.fit(X)

# transform the data to the new coordinate system
X_scaled = scaler.transform(X)

# create a PCA object with two components
pca = PCA(n_components=2)

# fit the PCA model to the scaled data
pca.fit(X_scaled)

# transform the scaled data to the new coordinate system
X_pca = pca.transform(X_scaled)

# plot the transformed data
for target, color in zip(targets, colors):
    indices = np.where(y == np.where(iris.target_names == target)[0][0])
    axes[1].scatter(X_pca[indices, 0], X_pca[indices, 1], c=color, label=target, alpha=1)

handles = []
for target, color in zip(targets, colors):
    handles.append(axes[1].scatter([], [], c=color, label=target))

axes[1].legend(handles=handles, loc='best')
axes[1].set_xlabel('First Principal Component')
```

```
axes[1].set_ylabel('Second Principal Component')

plt.show()

# get the principal components
pca.components_
```




```
array([[ 0.52106591, -0.26934744,  0.5804131 ,  0.56485654],
       [ 0.37741762,  0.92329566,  0.02449161,  0.06694199]])
```

The command `pc.components_` contains the matrix of principal components. Each row of this matrix corresponds to a principal component, displaying the weights of each feature of the data set.

Essentially, this means that the new variables, or principal components, are the following linear combinations of the main four features:

$$PC_1 = 0.52106591\,(\text{sepal length}) - 0.26934744\,(\text{sepal width}) + 0.5804131\,(\text{petal length}) + 0.56485654\,(\text{petal width})$$
$$PC_2 = 0.37741762\,(\text{sepal length}) + 0.92329566\,(\text{sepal width}) + 0.02449161\,(\text{petal length}) + 0.06694199\,(\text{petal width})$$

Note that since our original data is four-dimensional, it's impossible to plot it on the screen. Instead, and as a sample, we have plotted the scatter-plot of the first two dimensions, that is the sepal length and the sepal width. The interested reader can plot the scatter plot of the other dimensions as well, in a similar way. (Altogether, there are $\frac{4\times3}{2} = 6$ unique plots could be plotted between different pairs of features of the original data set.) While it's impossible to visualize the 4D data in one go, these 6 plots could give insights into the comparisons between pairs of features. It's more like describing different parts of the giant elephant in the room without being able to fully see or describe it in one go.

The second plot, on the other hand, captures *all* there is in the transformed, two-dimensional data. We can think of the second plot as capturing the most important variation patterns in the original, impossible-to-plot-in-one-go four-dimensional data set. That's where the power of PCA lies. In a way, it transforms the elephant in the room into something that can be described in one look and yet give us a pretty good understanding of the elephant.

Notes

[62]The image above is taken from this article: https://setosa.io/ev/principal-component-analysis/

[63]From Section 4.3.1 we remember that covariance measures how two variables vary together, meaning it quantifies the extent to which the values of two variables increase or decrease in tandem: if two variables have a positive covariance, it means that they tend to increase or decrease together. On the other hand, if two variables have a negative covariance, it means that they tend to vary in opposite directions. The covariance between two features is calculated as follows:

$$\sigma_{jk} = \frac{1}{n-1}\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)\left(x_{ik} - \bar{x}_k\right).$$

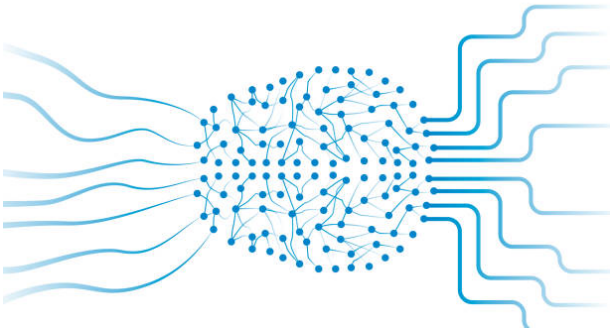
12.5 Further Reading

Here are a few references on PCA:

1. *Principal Component Analysis*, by Jolliffe, I. T. [18]. This book is a comprehensive treatment of PCA, including a discussion of the mathematical basis of the technique, practical issues in implementation, and applications to various fields.
2. "Principal component analysis", by Abdi, H., & Williams, L. J. [1]. This review article provides an overview of PCA, including its history, mathematical foundations, and applications. The article also discusses extensions of PCA, such as nonlinear PCA and sparse PCA.
3. "On lines and planes of closest fit to systems of points in space", by Pearson, K. [28]. This classic paper introduced the method of PCA and presented its application to the analysis of anthropometric data.
4. "Analysis of a complex of statistical variables into principal components", by Hotelling, H. [17]. This paper is a foundational work on PCA and provides a detailed discussion of the mathematical basis of the technique, including the derivation of the eigenvalue problem and the interpretation of the principal components.
5. "A tutorial on principal component analysis", by Shlens, J. [36]. This tutorial provides an accessible introduction to PCA, including a discussion of the mathematical basis of the technique, practical issues in implementation, and applications to image processing and computer vision.

Chapter 13

Neural Networks

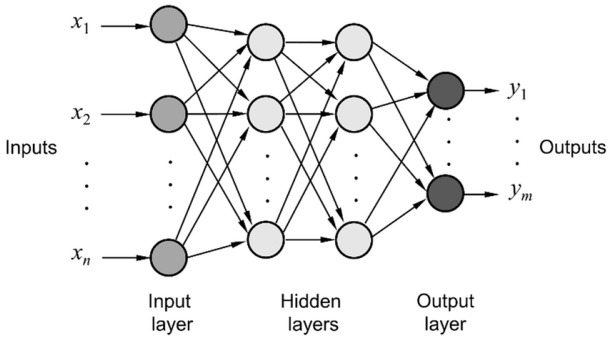


13.1 Introduction

Neural networks are computational systems inspired by the structure and function of the human brain. They are composed of multiple interconnected processing units that work together to solve complex problems. These units, also known as 'neurons', receive inputs, process them, and then generate outputs based on the strength of the connections between them. Neural networks have proven to be highly effective in a wide range of applications, from image and speech recognition to natural language processing and autonomous driving. They can be used for both regression and classification tasks.

At their core, neural networks rely on a mathematical model known as the 'artificial neuron'. An artificial neuron takes in one or more inputs and performs a weighted sum of these inputs, then applies a non-linear activation function to produce the output. This output is then passed on to the next layer of neurons until the final output is generated. The weights on the connections between neurons are adjusted during training so that the network can learn to produce the desired output for a given input.

Here's a schematic example of a multi-layer neural network:[64]



The training of a neural network typically involves the use of a large dataset of labeled examples. The network is presented with input-output pairs, and the weights on the connections between neurons are adjusted iteratively to minimize the difference between the network's output and the desired output. This process is known as 'backpropagation', and it allows the network to learn complex patterns and relationships in the data.

Despite their effectiveness, neural networks are not without their limitations. They can be computationally expensive to train and require a large amount of data to perform well. Additionally, the black box nature of neural networks can make it difficult to understand how they arrived at a particular decision or prediction.[65] However, ongoing research in the field is aimed at addressing these limitations and improving the performance and interpretability of neural networks.

In this chapter, we introduce neural networks, the concepts touched upon above, and their underlying mathematics.

13.2 Weights and Biases

Weights and biases are the basic and very important components of neural networks that play a crucial

role in determining the output of the network. In a neural network, a ***weight*** is a numerical value that is assigned to the connections between the neurons in the network. These weights determine the strength of the connections between the neurons and help to adjust the output of each neuron.

The weights in a neural network are adjusted during the training process using a technique called backpropagation (which we will discuss in depth later in the chapter). During backpropagation, the network compares its predicted output with the desired output and then adjusts the weights in order to reduce the difference between the two. This process is repeated multiple times until the network achieves the desired level of accuracy.

Biases, on the other hand, are like intercepts in linear regression. They are additional parameters that are added to the inputs of each neuron to control the output of the neuron. Biases help to shift the output of a neuron to the left or right, depending on the values of the input data.

Just like weights, biases are also adjusted during the training process using backpropagation. The goal of adjusting the biases is to ensure that the output of the neuron is properly calibrated to produce accurate predictions.

The next important concept in neural networks is that of an activation function.

13.3 Activation Functions

Activation functions are mathematical functions applied to the output of a neuron or a layer of neurons to introduce non-linearity into the network. They help to transform the input data into a form that can be more easily processed by the network.

Examples of activation functions include:

1. *Sigmoid*: This is a commonly used activation function that maps any input to a value between 0 and 1. It is given by the formula

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{13.1}$$

   We learned about this function in Section 8.2 when we learned about logistic regression. The sigmoid activation function is usually used in the output layer non-exclusive problems, such as multi-label classification problems where an example can belong to multiple classes.

2. *ReLU* (*Rectified Linear Unit*) function: This is another widely used activation function that maps any input to a value between 0 and infinity. It is given by the formula

$$f(x) = \max(0, x) \tag{13.2}$$

   ReLU function has become the most popular activation function for neural networks because of its simplicity and effectiveness. ReLU is computationally efficient to compute and it overcomes some of the limitations of the sigmoid function, such as the vanishing gradient problem.

3. *Tanh* (*Hyperbolic Tangent*) function: This activation function maps any input to a value between -1 and 1. It is given by the formula

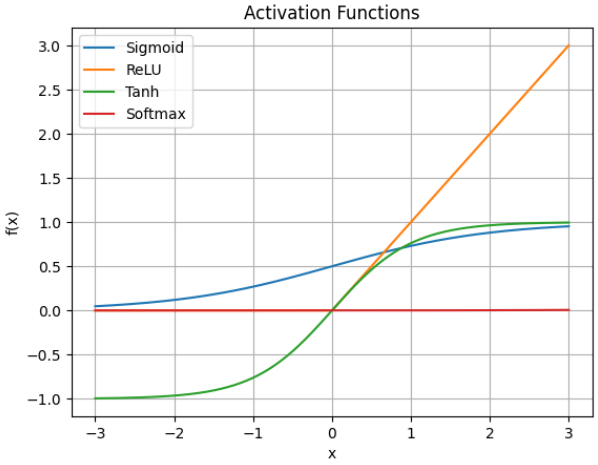
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{13.3}$$

4. *Softmax* function: This is an activation function that is often used in the output layer of a neural network for multi-class classification problems. It is given by the formula

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \text{ for } i = 1, \ldots, K \tag{13.4}$$

   for $i = 1, ..., K$, where K is the number of classes.

Here are the graphs of this functions:



Example 13.3.1. One example of a multi-label classification problem that uses the sigmoid activation function is image classification with multiple labels. For instance, in a medical imaging application, an image can be classified as having multiple types of abnormalities, such as "lung nodule" and "pleural effusion". We can train a neural network with a sigmoid activation function in the output layer to produce a probability for each label.

Suppose we have K different labels, denoted by $y_1, y_2, \ldots, y_K$. Given an input image, the output of

the neural network for label k can be represented as follows:

$$p_k = \sigma(z_k) = \frac{1}{1+e^{-z_k}}$$

where z_k is the logit (output of the previous layer before the activation function) for label k, and σ is the sigmoid function.

The output of the neural network for the entire set of K labels can be represented as a vector:

$$\mathbf{p} = \begin{bmatrix} p_1 & p_2 & \cdots & p_K \end{bmatrix}$$

where each element p_k represents the probability of the input image belonging to label y_k.

Example 13.3.2. Now suppose we have a dataset of handwritten digits from 0 to 9 and we want to classify them into different categories. Since each image contains only one digit, this is a multi-class classification problem where classes are mutually exclusive.

We can use a neural network with a softmax activation function in the output layer to predict the probability of each digit category for each input image. The output of the network for a given image can be represented as follows:

$$\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_9 \end{bmatrix} = \begin{bmatrix} \frac{e^{z_0}}{\sum_{i=0}^{9} e^{z_i}} \\ \frac{e^{z_1}}{\sum_{i=0}^{9} e^{z_i}} \\ \vdots \\ \frac{e^{z_9}}{\sum_{i=0}^{9} e^{z_i}} \end{bmatrix}$$

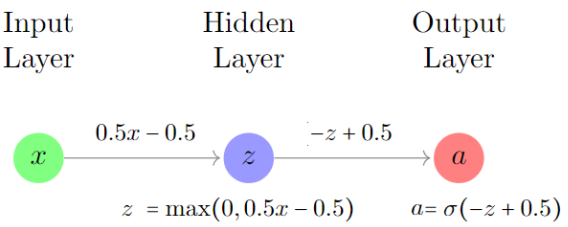
where p_0, p_1, ..., p_9 are the predicted probabilities for each digit category, e^{z_0}, e^{z_1}, ..., e^{z_9} are the exponentiated logits (outputs of the previous layer before the activation function) for each category.

Example 13.3.3. Let's consider a neural network with a single input neuron, a single hidden neuron with ReLU activation function, and a single output neuron with sigmoid activation function. The weights and biases are specified as follows:

- Weight w_1 connects the input neuron to the hidden neuron, with a weight value of 0.5.
- Weight w_2 connects the hidden neuron to the output neuron, with a weight value of −1.
- Bias b_1 is added to the hidden neuron, with a bias value of −0.5.
- Bias b_2 is added to the output neuron, with a bias value of 0.5.

$$z = \max(0, w_1 x + b_1) = \max(0, 0.5x - 0.5)$$

$$a = \sigma(w_2 z + b_2) = \sigma(-z + 0.5) = \frac{1}{1+e^{z-0.5}}$$



13.4 Cost Functions

The ***cost function***, also known as the ***loss function***, measures how well the model is performing on the given training data. The choice of the cost function depends on the type of problem, such as classification or regression, and the output layer activation function.

For classification problems, where the output layer activation function is typically sigmoid or softmax, the most commonly used cost function is binary cross-entropy for binary classification and categorical cross-entropy for multi-class classification. For regression problems, where the output layer activation function is typically linear, the most commonly used cost function is mean squared error (MSE). These are defined as follows:

1. *Binary cross-entropy* for binary classification:

$$L(\hat{y}, y) = -\frac{1}{m}\sum_{i=1}^{m}(y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})) \tag{13.5}$$

   where $\hat{y}$ is the predicted output, y is the true label, and m is the number of samples in the training data.

2. *Categorical cross-entropy* for multi-class classification:

$$L(\hat{y}, y) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)}\log(\hat{y}_k^{(i)}) \tag{13.6}$$

   where $\hat{y}$ is the predicted output, y is the true label, m is the number of samples in the training data, and K is the number of classes.

3. *Mean squared error* for regression:

$$L(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2 \tag{13.7}$$

   where $\hat{y}$ is the predicted output, y is the true output, and m is the number of samples in the training data.

Here's an example of binary cross-entropy for binary classification with 10 data points:

Example 13.4.1. Suppose we have a binary classification problem with two classes: 0 and 1. Let $\hat{y}_i$ be the predicted probability that the ith input belongs to class 1, and let y_i be the true label (either 0 or 1) for the ith input.

Suppose we have the following data points:

Input	**True label** (y)
x_1	1
x_2	1
x_3	0
x_4	0
x_5	1
x_6	0
x_7	0
x_8	1
x_9	1
x_{10}	0

Suppose our model predicts the following probabilities for each input:

Input	**Predicted probability** $(\hat{y})$
x_1	0.9
x_2	0.8
x_3	0.3
x_4	0.2
x_5	0.7
x_6	0.4
x_7	0.1
x_8	0.6
x_9	0.8
x_{10}	0.2

Then the binary cross-entropy loss for the entire dataset is:

$$\frac{1}{10}\left[1\log(0.9) + 1\log(0.8) + 0\log(0.3) + \dots\right] = 0.2961$$

So:

$$L(\hat{y}_i, y_i) = 0.2961$$

During training, the goal would be to minimize this overall loss with respect to the model parameters.

13.5 Backpropagation

Backpropagation is an algorithm used to train neural networks by computing the gradient of the loss function with respect to the weights and biases of the network. The gradient can then be used to update the weights and biases using gradient descent to minimize the loss function.

Intuitively, backpropagation works by propagating the error from the output layer back through the network to the input layer, calculating the contribution of each neuron to the error. This contribution is then used to update the weights and biases of the neurons so that the overall error is reduced in the next iteration of training.

The backpropagation algorithm involves two main steps: the *forward pass* and the *backward pass*.

1. ***Forward pass:*** during the forward pass, the input data is fed into the neural network, and the output is calculated layer by layer, using the activation function of each neuron. The output of the network is then compared to the actual output, and the loss function is computed.
2. ***Backward pass:*** during the backward pass, the error in the output layer is propagated back through the network, layer by layer, to compute the gradient of the loss function with respect to the weights and biases. This is done using the chain rule of calculus, which allows us to compute the derivative of a composite function.

By iteratively computing the forward pass and backward pass, and updating the weights and biases with the gradient descent algorithm, backpropagation can be used to train a neural network to minimize the loss function.

The formulas for backpropagation are as follows:

1. Forward pass:

$$z^{(l)} = w^{(l)}a^{(l-1)} + b^{(l)} \tag{13.8}$$

$$a^{(l)} = g(z^{(l)}) \tag{13.9}$$

   where $z^{(l)}$ is the input to the activation function g in layer l, $w^{(l)}$ and $b^{(l)}$ are the weights and biases of layer l, and $a^{(l-1)}$ is the output of layer $l-1$.
2. Backward pass:

$$\delta^{(L)} = \nabla_{a^{(L)}} J(\theta) \odot g'(z^{(L)}) \tag{13.10}$$

$$\delta^{(l)} = ((w^{(l+1)})^T \delta^{(l+1)}) \odot g'(z^{(l)}) \tag{13.11}$$

$$\frac{\partial J}{\partial w^{(l)}} = \delta^{(l)}(a^{(l-1)})^T \tag{13.12}$$

$$\frac{\partial J}{\partial b^{(l)}} = \delta^{(l)} \tag{13.13}$$

   where $\delta^{(L)}$ is the error in the output layer, $\nabla_{a^{(L)}} J(\theta)$ is the gradient of the loss function with respect to the output (more on this below), $\odot$ represents element-wise multiplication,

also known as the *Hadamard product* and g' is the derivative of the activation function.[66]

The first equation (13.10) calculates the error at the output layer. The second equation (13.11) calculates the error for each layer by backpropagating the error from the next layer. The last two equations (13.12-13.13) calculate the gradient of the loss function with respect to the weights and biases, which can then be used to update the weights and biases using gradient descent.

Note that the error in the output layer is calculated using the Hadamard product because we need to take into account the contribution of each output neuron to the overall error. Each output neuron's error is calculated by taking the difference between its predicted output and the true output, and multiplying it by the derivative of the activation function with respect to its input. This derivative term represents the sensitivity of the output to changes in the input, and by multiplying it with the error term we obtain the contribution of that neuron to the overall error.

We can then update the errors using the gradient descent algorithm. Specifically, the weights and biases are updated according to the following formulas:

$$w_{i,j}^{(l)} = w_{i,j}^{(l)} - \alpha \frac{\partial J}{\partial w_{i,j}^{(l)}} \tag{13.14}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial J}{\partial b_i^{(l)}} \tag{13.15}$$

where α is the learning rate, J is the cost function, and $\frac{\partial J}{\partial w_{i,j}^{(l)}}$ and $\frac{\partial J}{\partial b_i^{(l)}}$ are the gradients of the cost function with respect to the weights and biases, respectively. These gradients are computed using backpropagation, as I explained earlier. The gradient descent algorithm iteratively updates the weights and biases until the cost function is minimized or a stopping criterion is met.

Remark 13.5.1. Remember from the chapter on Calculus that the gradient of the cost function with respect to the activations of the last layer, denoted by $\nabla_{a^{(L)}} J(\theta)$, is calculated as follows:

$$\nabla_{a^{(L)}} J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial a_1^{(L)}} & \frac{\partial J(\theta)}{\partial a_2^{(L)}} & \cdots & \frac{\partial J(\theta)}{\partial a_K^{(L)}} \end{bmatrix} \tag{13.16}$$

where K is the number of neurons in the last layer.

The partial derivative of the cost function with respect to each activation in the last layer is calculated using the chain rule:

$$\frac{\partial J(\theta)}{\partial a_k^{(L)}} = \sum_{i=1}^{K} \frac{\partial J(\theta)}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial a_k^{(L)}} \tag{13.17}$$

Here, $z_i^{(L)}$ is the weighted sum of inputs to neuron i in the last layer, and $a_k^{(L)}$ is the activation of neuron k in the last layer. The first term in the right-hand side of the equation is the derivative of the cost function with respect to the weighted sum of inputs to each neuron in the last layer, and can be calculated using the backpropagation algorithm. The second term is the derivative of the weighted sum of inputs with respect to the activation of each neuron in the last layer, which is simply the weight connecting the neuron to the previous layer. Therefore, we have:

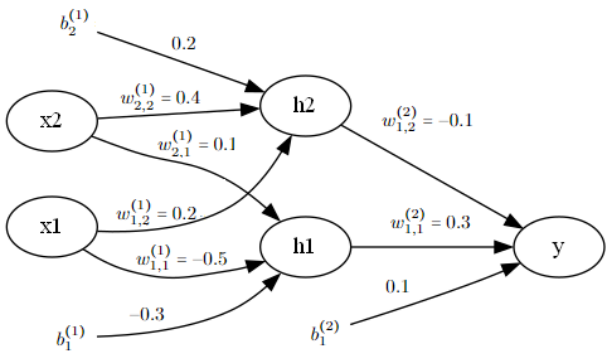
$$\frac{\partial J(\theta)}{\partial a_k^{(L)}} = \sum_{i=1}^{K} \frac{\partial J(\theta)}{\partial z_i^{(L)}} w_{i,k}^{(L)} \tag{13.18}$$

This gives us the gradient of the cost function with respect to the activations in the last layer.

Example 13.5.1. Consider a neural network with one hidden layer, with two nodes per input and the hidden layer, and one output layer, for performing a regression task. Suppose the weights and biases are initialized randomly as follows:

$$w_{1,1}^{(1)} = -0.5, \quad w_{1,2}^{(1)} = 0.2, \quad b_1^{(1)} = -0.3$$
$$w_{2,1}^{(1)} = 0.1, \quad w_{2,2}^{(1)} = 0.4, \quad b_2^{(1)} = 0.2$$
$$w_{1,1}^{(2)} = 0.3, \quad w_{1,2}^{(2)} = -0.1, \quad b_1^{(2)} = 0.1$$

Here's an abstract illustration of our neural network:



To illustrate how the neural networks work, let's use the input $x_1 = 0.2$ and $x_2 = 0.4$ to compute the output and the gradients.

Here are the steps to perform forward propagation:

1. Calculation of the input values of the hidden

layer:

$$z_1 = w_{1,1}^{(1)} x_1 + w_{2,1}^{(1)} x_2 + b_1^{(1)} =$$
$$-0.5(0.2) + 0.1(0.4) - 0.3 = -0.38$$
$$z_2 = w_{1,2}^{(1)} x_1 + w_{2,2}^{(1)} x_2 + b_2^{(1)} =$$
$$0.2(0.2) + 0.4(0.4) + 0.2 = 0.34$$

2. Application of the activation function to get the output values of the hidden layer:

$$h_1 = \sigma(z_1) = \sigma(-0.38) \approx 0.41$$
$$h_2 = \sigma(z_2) = \sigma(0.34) \approx 0.59$$

where σ is the sigmoid function

3. Calculation of the input value of the output layer:

$$z_3 = w_{1,1}^{(2)} h_1 + w_{1,2}^{(2)} h_2 + b_1^{(2)} =$$
$$0.3(0.41) - 0.1(0.59) + 0.1 \approx 0.08$$

4. Application of the activation function to get the output value of the output layer:

$$\hat{y} = \sigma(z_3) = \sigma(0.08) \approx 0.52$$

So, for these input values, the output value of the neural network is approximately 0.52.

Backward propagation can then be performed using the formulas (13.10 - 13.13), and the weights and biases can be improved using gradient descent (13.14-13.15). I will leave this to the reader.

13.6 Python Implementations

We now provide a simple example of a neural network using TensorFlow's Keras library for a classification problem. We use Tensorflow and Keras.

TensorFlow is a popular open-source machine-learning library developed by Google. It provides a flexible and powerful platform for building and training various types of machine learning models. TensorFlow is used extensively in both academia and industry for research and production-level applications.

Keras, on the other hand, is a high-level neural network API that runs on top of TensorFlow (and other deep learning libraries). It provides a simplified interface for building and training neural networks. Keras is known for its ease of use, flexibility, and modularity, making it a popular choice among researchers and developers.

In this code, we use TensorFlow and Keras to build and train a neural network model for a classification problem. We use TensorFlow as the backend engine for Keras, allowing us to take advantage of TensorFlow's powerful computation capabilities.

Unlike the previous chapters where I mostly gave the code in one chunk, I think it's valuable to write down our code here in several steps, where each step is clearly explained.

First, let's import the necessary libraries:

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from sklearn import datasets
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
```

Here is an explanation of what these libraries are:

- `Sequential` is a class in Keras that allows us to create a neural network model where layers are added one after another.
- `Dense` is another class in Keras that allows us to create a fully connected layer of neurons.
- `Activation` is a class in Keras that provides a variety of activation functions that can be applied to the output of a layer.
- `EarlyStopping` is a class in Keras that allows us to stop the training process early if the validation loss stops improving.
- `MinMaxScaler` is a class in scikit-learn that scales the features of the dataset between 0 and 1.

We then load the iris dataset using scikit-learn's built-in dataset, and split the iris dataset into training and testing sets, where 70% of the data is used for training and 30% for testing.

```
# load the data
iris = datasets.load_iris()
# Split data into training and testing sets
X_train, X_test, y_train, y_test =  train_test_split(iris.data, iris.target,
test_size=0.3)
```

Next, we normalize the features of the data using MinMaxScaler. This scales the features such that they lie between 0 and 1.

```
# normalize the data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

We define a sequential neural network model using Keras. The model consists of three dense layers, where the first two have 16 neurons and use the ReLU activation function, and the output layer has 3 neurons (one for each class) and uses the softmax activation function.endnoteRecall that iris has three target classes, 'setosa', 'versicolor' and 'virginica'. You can see this using this code: `iris.target_names`.

The choice of the number of neurons in the hidden layers and the activation functions depends on the complexity of the problem, the amount and quality of the data available, and the experience of the practitioner. In this code, the neural network has two hidden layers, each with 16 neurons and the ReLU activation function.

The output layer has 3 neurons, one for each class in the iris dataset, and uses the softmax activation function. We already know that Softmax is a popular activation function used for multi-class classification problems. It transforms the output of the final layer into a probability distribution over the classes, ensuring that the sum of the probabilities for all classes is 1. Softmax allows us to interpret the output of the neural network as the probability that the input belongs to each of the classes.

```
# define the model
model = Sequential()
model.add(Dense(16, input_dim=4))
model.add(Activation('relu'))
model.add(Dense(16))
model.add(Activation('relu'))
model.add(Dense(3))
model.add(Activation('softmax'))
```

We then set up early stopping to prevent overfitting. The early stopping will monitor the validation loss, and if it does not improve for 5 consecutive epochs, the training will stop early.

Overfitting occurs when a machine learning model is too complex, and it starts to learn the noise and peculiarities of the training data instead of the underlying pattern that generalizes to new data. In other words, the model becomes too specialized to the training data and fails to perform well on new data.

Early stopping is a technique to prevent overfitting in deep learning models. It works by monitoring the validation loss during training and stopping the training process when the validation loss stops improving. The idea is that as the model learns the training data better and better, it should also perform better on the validation set. However, after a certain point, the model starts to overfit, and the validation loss starts to increase. At this point, early stopping kicks in and stops the training process before the model becomes too specialized to the training data.

In the code, we set up early stopping by creating an EarlyStopping object and passing it to the fit method as a callback. The EarlyStopping object monitors the validation loss by specifying `monitor='val_loss'`. If the validation loss does not improve for 5 consecutive epochs, as specified by `patience=5`, the training will stop early. This

means that the weights of the model at the end of the epoch with the lowest validation loss will be used for prediction. The goal of early stopping is to prevent overfitting and improve the generalization of the model to new data.

```
# set up early stopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
```

We then compile the model, specifying the loss function, optimizer, and metrics to use during training. In this case, we are using the sparse categorical cross-entropy loss function, the Adam optimizer, and accuracy as a metric to evaluate the model.

```
# compile the model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

Finally, we train the model on the training set, specifying the number of epochs and the batch size. We also use early stopping to prevent overfitting. The model's history is returned in the history variable. The validation_split argument specifies that 20% of the training data should be used for validation during training.

```
# train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50, batch_size=8,
callbacks=[early_stop])
```

This is what my system outputs at this stage:

```
Epoch 1/50
11/11 [=========] - 1s 18ms/step - loss: 1.1414 - accuracy: 0.3095 - val_loss: 1.1402 - val_accuracy: 0.3810
Epoch 2/50
11/11 [=========] - 0s 4ms/step - loss: 1.1095 - accuracy: 0.3095 - val_loss: 1.1130 - val_accuracy: 0.3810
Epoch 3/50
11/11 [=========] - 0s 4ms/step - loss: 1.0822 - accuracy: 0.5119 - val_loss: 1.0871 - val_accuracy: 0.6667
Epoch 4/50
11/11 [=========] - 0s 4ms/step - loss: 1.0534 - accuracy: 0.7024 - val_loss: 1.0606 - val_accuracy: 0.6667
Epoch 5/50
11/11 [=========] - 0s 4ms/step - loss: 1.0235 - accuracy: 0.7024 - val_loss: 1.0294 - val_accuracy: 0.6667
Epoch 6/50
11/11 [=========] - 0s 4ms/step - loss: 0.9934 - accuracy: 0.7024 - val_loss: 1.0003 - val_accuracy: 0.6667
Epoch 7/50
11/11 [=========] - 0s 4ms/step - loss: 0.9575 - accuracy: 0.7024 - val_loss: 0.9680 - val_accuracy: 0.6667
Epoch 8/50
11/11 [=========] - 0s 4ms/step - loss: 0.9236 - accuracy: 0.7024 - val_loss: 0.9337 - val_accuracy: 0.6667
Epoch 9/50
11/11 [=========] - 0s 4ms/step - loss: 0.8877 - accuracy: 0.7024 - val_loss: 0.8981 - val_accuracy: 0.6667
Epoch 10/50
11/11 [=========] - 0s 4ms/step - loss: 0.8543 - accuracy: 0.7024 - val_loss: 0.8637 - val_accuracy: 0.6667
Epoch 11/50
11/11 [=========] - 0s 4ms/step - loss: 0.8147 - accuracy: 0.7024 - val_loss: 0.8259 - val_accuracy: 0.6667
Epoch 12/50
11/11 [=========] - 0s 4ms/step - loss: 0.7781 - accuracy: 0.7024 - val_loss: 0.7880 - val_accuracy: 0.6667
Epoch 13/50
11/11 [=========] - 0s 4ms/step - loss: 0.7418 - accuracy: 0.7024 - val_loss: 0.7503 - val_accuracy: 0.6667
Epoch 14/50
11/11 [=========] - 0s 4ms/step - loss: 0.7040 - accuracy: 0.7024 - val_loss: 0.7153 - val_accuracy: 0.6667
Epoch 15/50
11/11 [=========] - 0s 5ms/step - loss: 0.6669 - accuracy: 0.7024 - val_loss: 0.6771 - val_accuracy: 0.6667
Epoch 16/50
11/11 [=========] - 0s 4ms/step - loss: 0.6327 - accuracy: 0.7024 - val_loss: 0.6372 - val_accuracy: 0.6667
Epoch 17/50
```

```
11/11 [=========] - 0s 4ms/step - loss: 0.5981 - accuracy: 0.7024 - val_loss: 0.6105 - val_accuracy: 0.6667
Epoch 18/50
11/11 [=========] - 0s 4ms/step - loss: 0.5682 - accuracy: 0.7024 - val_loss: 0.5839 - val_accuracy: 0.6667
Epoch 19/50
11/11 [=========] - 0s 4ms/step - loss: 0.5413 - accuracy: 0.7024 - val_loss: 0.5591 - val_accuracy: 0.6667
Epoch 20/50
11/11 [=========] - 0s 4ms/step - loss: 0.5182 - accuracy: 0.7024 - val_loss: 0.5346 - val_accuracy: 0.6667
Epoch 21/50
11/11 [=========] - 0s 4ms/step - loss: 0.4958 - accuracy: 0.7024 - val_loss: 0.5137 - val_accuracy: 0.7143
Epoch 22/50
11/11 [=========] - 0s 4ms/step - loss: 0.4760 - accuracy: 0.7024 - val_loss: 0.4958 - val_accuracy: 0.7143
Epoch 23/50
11/11 [=========] - 0s 4ms/step - loss: 0.4582 - accuracy: 0.7262 - val_loss: 0.4780 - val_accuracy: 0.7143
Epoch 24/50
11/11 [=========] - 0s 4ms/step - loss: 0.4421 - accuracy: 0.7262 - val_loss: 0.4613 - val_accuracy: 0.7619
Epoch 25/50
11/11 [=========] - 0s 4ms/step - loss: 0.4247 - accuracy: 0.7500 - val_loss: 0.4509 - val_accuracy: 0.7619
Epoch 26/50
11/11 [=========] - 0s 4ms/step - loss: 0.4111 - accuracy: 0.7619 - val_loss: 0.4395 - val_accuracy: 0.7619
Epoch 27/50
11/11 [=========] - 0s 4ms/step - loss: 0.3970 - accuracy: 0.7976 - val_loss: 0.4275 - val_accuracy: 0.7619
Epoch 28/50
11/11 [=========] - 0s 4ms/step - loss: 0.3849 - accuracy: 0.8214 - val_loss: 0.4175 - val_accuracy: 0.7619
Epoch 29/50
11/11 [=========] - 0s 4ms/step - loss: 0.3715 - accuracy: 0.9048 - val_loss: 0.4008 - val_accuracy: 0.8095
Epoch 30/50
11/11 [=========] - 0s 4ms/step - loss: 0.3583 - accuracy: 0.9286 - val_loss: 0.3927 - val_accuracy: 0.8095
Epoch 31/50
11/11 [=========] - 0s 4ms/step - loss: 0.3495 - accuracy: 0.9524 - val_loss: 0.3796 - val_accuracy: 0.8095
Epoch 32/50
11/11 [=========] - 0s 4ms/step - loss: 0.3366 - accuracy: 0.9524 - val_loss: 0.3780 - val_accuracy: 0.8095
Epoch 33/50
11/11 [=========] - 0s 4ms/step - loss: 0.3264 - accuracy: 0.9405 - val_loss: 0.3623 - val_accuracy: 0.8571
Epoch 34/50
11/11 [=========] - 0s 4ms/step - loss: 0.3148 - accuracy: 0.9643 - val_loss: 0.3548 - val_accuracy: 0.8571
Epoch 35/50
11/11 [=========] - 0s 4ms/step - loss: 0.3024 - accuracy: 0.9524 - val_loss: 0.3510 - val_accuracy: 0.8095
Epoch 36/50
11/11 [=========] - 0s 4ms/step - loss: 0.2957 - accuracy: 0.9524 - val_loss: 0.3420 - val_accuracy: 0.8571
Epoch 37/50
11/11 [=========] - 0s 4ms/step - loss: 0.2822 - accuracy: 0.9762 - val_loss: 0.3297 - val_accuracy: 0.9048
Epoch 38/50
11/11 [=========] - 0s 4ms/step - loss: 0.2752 - accuracy: 0.9762 - val_loss: 0.3208 - val_accuracy: 0.9524
Epoch 39/50
11/11 [=========] - 0s 4ms/step - loss: 0.2640 - accuracy: 0.9762 - val_loss: 0.3194 - val_accuracy: 0.8571
Epoch 40/50
11/11 [=========] - 0s 4ms/step - loss: 0.2608 - accuracy: 0.9643 - val_loss: 0.3155 - val_accuracy: 0.8571
Epoch 41/50
11/11 [=========] - 0s 4ms/step - loss: 0.2470 - accuracy: 0.9762 - val_loss: 0.3027 - val_accuracy: 0.9524
Epoch 42/50
11/11 [=========] - 0s 4ms/step - loss: 0.2415 - accuracy: 0.9762 - val_loss: 0.2932 - val_accuracy: 0.9524
Epoch 43/50
11/11 [=========] - 0s 4ms/step - loss: 0.2354 - accuracy: 0.9762 - val_loss: 0.2897 - val_accuracy: 0.9524
Epoch 44/50
11/11 [=========] - 0s 4ms/step - loss: 0.2245 - accuracy: 0.9762 - val_loss: 0.2832 - val_accuracy: 0.9524
Epoch 45/50
11/11 [=========] - 0s 4ms/step - loss: 0.2179 - accuracy: 0.9762 - val_loss: 0.2783 - val_accuracy: 0.9524
Epoch 46/50
11/11 [=========] - 0s 4ms/step - loss: 0.2130 - accuracy: 0.9762 - val_loss: 0.2740 - val_accuracy: 0.9524
Epoch 47/50
11/11 [=========] - 0s 4ms/step - loss: 0.2041 - accuracy: 0.9762 - val_loss: 0.2685 - val_accuracy: 0.9524
Epoch 48/50
11/11 [=========] - 0s 4ms/step - loss: 0.1994 - accuracy: 0.9762 - val_loss: 0.2670 - val_accuracy: 0.9524
Epoch 49/50
11/11 [=========] - 0s 4ms/step - loss: 0.1928 - accuracy: 0.9762 - val_loss: 0.2551 - val_accuracy: 0.9524
Epoch 50/50
11/11 [=========] - 0s 4ms/step - loss: 0.1910 - accuracy: 0.9762 - val_loss: 0.2521 - val_accuracy: 0.9524
```

The output is the training log of the machine-learning model. Specifically, it shows the performance of the model on the training set and a validation set over multiple epochs. Here is a breakdown of the different columns:

- Epoch: The current epoch number. An *epoch* refers to a single pass through the entire dataset during the training phase of a neural network model. In other words, an epoch is a complete iteration of the training process where the model receives input data, makes predictions, calculates the loss, and updates its parameters based on the error.
- 11/11: The number of batches processed in the current epoch and the total number of batches in the training set.
- loss: The loss (i.e., error) of the model on the training set for the current epoch. As we already know, the loss is a measure of how well the model's predictions match the ground truth labels in the training set. It represents the difference between the predicted values and the true values of the target variable. During the training process, the goal is to minimize the loss function to improve the model's accuracy.
- accuracy: The accuracy of the model on the training set for the current epoch. We know that accuracy is a measure of how well the model's predictions match the true labels, expressed as a percentage. It is calculated as the number of correct predictions divided by the total number of predictions. During training, the goal is to maximize accuracy to improve the model's performance.
- val_loss: The loss of the model on the validation set for the current epoch. Val_loss is the value of the loss function on the validation dataset, which is a separate dataset used to evaluate the model's performance during the training phase. This metric is used to prevent overfitting, which occurs when the model performs well on the training data but poorly on new, unseen data.
- val_accuracy: The accuracy of the model on the validation set for the current epoch. Val_accuracy is the accuracy metric calculated on a validation dataset. It measures how well the model generalizes to new, unseen data and is used to evaluate the model's performance during training.

The model was trained for 50 epochs, and its accuracy on the training set appears to be improving, as it increases from 0.3095 in epoch 1 to 0.7024 in epoch 20, and converges to 0.9762 until the end. The accuracy on the validation set also improves, from 0.3810 in epoch 1 to 0. in epoch 3, and stays at 0.9762 until the end, starting epoch 37. This suggests the model is learning from the training set and generalizing to the validation set.

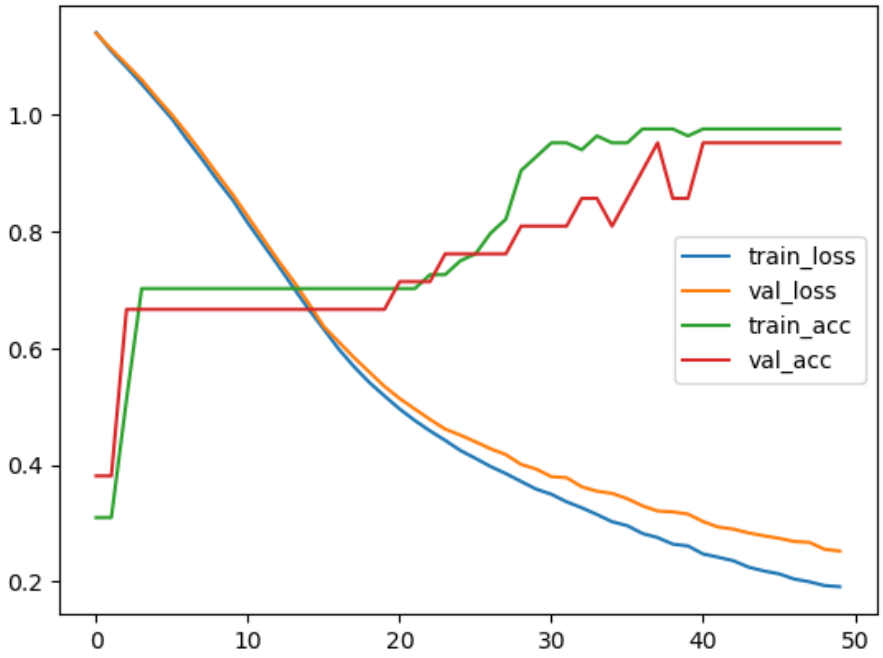
In the given code, the validation set is created using the `validation_split` parameter of the `fit` function. Specifically, `validation_split=0.2` means that 20% of the training data will be used as a validation set.

During the training process, the model is trained on the remaining 80% of the training data and evaluated on the validation set after each epoch. The purpose of the validation set is to monitor the performance of the model during training and to prevent overfitting by stopping the training process early if the validation loss starts to increase.

We now plot the evaluation functions:

```
# plot the model history
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.plot(history.history['accuracy'], label='train_acc')
plt.plot(history.history['val_accuracy'], label='val_acc')
plt.legend()
plt.show()
```

This is what we'll get with:



A higher validation loss compared to the training loss typically indicates that the model is overfitting the training data, meaning that it performs well on the training data but poorly on new, unseen data (such as the validation data).

Overfitting occurs when the model learns the noise in the training data, rather than the underlying patterns that generalize to new data. This results in a model that performs well on the training data but poorly on the validation or test data. In other words, the model is memorizing the training data instead of learning from it.

In contrast, if the validation loss is lower than the training loss, it may indicate that the model is *underfitting*, meaning that it is too simple and does not capture the complexity of the data. Similarly, when `train_acc` stays on top of `val_acc`, it means that the model is overfitting to the training data.

It is important to monitor both `train_acc` and `val_acc` during training to ensure that the model is not overfitting. If the gap between `train_acc` and `val_acc` is large, regularization techniques or other strategies may be necessary to improve the model's performance on the validation or test data.

The fact that the training loss and validation loss match up almost perfectly until the middle of training indicates that the model is not overfitting the training data during the initial phases of training. This is a good sign, as it means the model is not simply memorizing the training data and is learning more generalizable patterns.

However, the fact that there is some overfitting happening after the middle of training indicates that the model is starting to memorize the training data and may be becoming too specialized to the training set. This is a common problem in machine learning and can lead to poor performance on new, unseen data.

The fact that the training accuracy and validation accuracy evolve closely together throughout the training is also a good sign, as it suggests that the model is not simply memorizing the training data and is generalizing well to new data. However, the exceptions in around two-thirds of the training could be an indication of overfitting or other issues.

Overall, it seems that the model is performing well but could benefit from some additional regularization techniques to prevent overfitting and improve generalization to new data. *regularization* is a technique that adds a penalty term to the loss function during training to prevent the model from overfitting the training data. There are several types of regularization, such as L1 and L2 regularization, dropout, and data augmentation.

We're not going to do anything else to avoid overfitting because the model, overall, seems to be performing well. In fact, this can be seen more accurately in the final step, which is to evaluate the model. We use the following codes for this:

```
# evaluate the model
test_loss, test_acc = model.evaluate(X_test, y_test)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
```

Which outputs the following:

```
2/2 [==============================] - 0s 3ms/step - loss: 0.3637 - accuracy: 0.9111
Test Loss: 0.3636517822742462
Test Accuracy: 0.9111111164093018
```

This output is the result of evaluating a neural network model on a test dataset. The model achieved a test loss of 0.3637, which is a measure of the difference between the predicted output and the actual output. A lower test loss indicates that the model's predictions are closer to the actual output, which is desirable.

The model also achieved a test accuracy of 0.9111, which means that it correctly classified 91.11% of the test dataset. Accuracy is a measure of how well the model's predictions match the actual labels, with a higher accuracy indicating better performance.

Overall, these results suggest that the model is performing relatively well on the test dataset and is likely to generalize well to new, unseen data. However, it is important to note that evaluating a model on a single test dataset may not be sufficient to fully assess its performance, and additional testing and validation may be necessary.

Notes

[64]The image credit goes to this paper: https://journals.sagepub.com/doi/10.4137/BECB.S31601

[65]In the context of neural networks, the term "black box nature" refers to the inability to understand the internal workings of the neural network. A neural network can be seen as a complex system that takes in inputs and produces outputs through a series of mathematical operations. However, the specific steps taken by the neural network to arrive at a particular output are often difficult to interpret or explain. This lack of transparency can be a problem in some applications, such as in fields like medicine or law, where it is important to understand how a decision was made. The black box nature of neural networks means that it can be challenging to diagnose errors or biases in the network's decisions, and it can also make it difficult to improve or modify the network's behavior.

[66]Here is a simple example. Suppose we have two matrices: $A = \begin{bmatrix} 2 & 4 & 6 & 8 & 10 & 12 \end{bmatrix}$ and $B = \begin{bmatrix} 1 & 3 & 5 & 7 & 9 & 11 \end{bmatrix}$

Then: $A \odot B = \begin{bmatrix} 2 \times 1 & 4 \times 3 & 6 \times 5 & 8 \times 7 & 10 \times 9 & 12 \times 11 \end{bmatrix} = \begin{bmatrix} 2 & 12 & 30 & 56 & 90 & 132 \end{bmatrix}$.

13.7 Further Reading

There are many amazing resources on deep learning and neural networks, including open-source ones. Here are some:

1. *Deep Learning*, by Ian Goodfellow, Yoshua Bengio, and Aaron Courville [12]. This book provides a comprehensive introduction to deep learning, including the underlying mathematical concepts and practical implementation details. It covers topics such as feedforward neural networks, convolutional neural networks, recurrent neural networks, optimization algorithms, and more.
2. *Neural Networks and Deep Learning*, by Michael Nielsen [27]. This book has great discussions of different concepts in deep learning, particularly backpropagation (Chapter 2). The book is freely available to everyone.
3. *Neural Networks and Deep Learning: A Textbook*, by Charu Aggarwal [2]. This textbook covers the basics of neural networks and deep learning, including mathematical concepts, optimization algorithms, and popular architectures such as convolutional and recurrent neural networks. It also provides practical examples and Python code for implementing these models.
4. *Dive into Deep Learning*, by Zang, et.al [45]. This book is a true gem when it comes to deep learning, and I cannot emphasize enough checking it out. The book dives deep into different neural network architectures, their mathematics and implementations in various major frameworks, such as PyTorch and TensorFlow. The book is open source and available to everyone in the public.

Chapter 14

Coda

In this book, we covered a relatively extensive introduction to most of the core topics and concepts in the mathematics of data science—both on the pure math front and on the applied front where machine learning algorithms are at stake.

But the journey doesn't end here. There are more advanced and sophisticated things to say about every single thing discussed in this book. For anyone who is interested in deeper dives into any of the topics covered, most of the books cited in the 'Further Reading' sections of the chapters would serve well.

But if you would like to keep up with cutting-edge advances, you have got to check out arXiv, at the address of https://arxiv.org. Quoting from the main page of the website, "arXiv is a free distribution service and an open-access archive for 2,246,568 scholarly articles in the fields of physics, mathematics, computer science, quantitative biology, quantitative finance, statistics, electrical engineering and systems science, and economics. Materials on this site are not peer-reviewed by arXiv." Related to data science, many if not all of the recent advances, both on the theoretical and the applied front in the field, appear here even before getting officially published in academic journals or companies' web pages.

As far as mathematics in data science is concerned, one thing that I would've liked to cover in this book, but couldn't due to scope limits, was the topic of 'Graph Neural Networks' (GNN), also known as 'Geometric Deep Learning' (GDL), which hinges on more abstract and advanced mathematics, such as abstract algebra, graph theory and manifold geometry. Geometric deep learning is an emerging field at the intersection of deep learning and geometry, which aims to develop deep learning algorithms capable of operating directly on non-Euclidean structured data, such as graphs and manifolds.

The key idea behind GDL is to exploit the underlying geometric structure of data, such as the graph structure of social networks, molecular structures, or brain connectivity graphs, to enhance the performance of deep learning models. GDL has a wide range of applications in various fields, including computer vision, natural language processing, drug discovery, and social network analysis, among others.

GDL is a relatively new and specialized field, so it may be considered advanced compared to more traditional deep learning methods. Aside from a growing number of research papers, I'm aware of a few resources on the topic, which are fantastic on their own. They can be found at the following addresses:

- "A Gentle Introduction to Graph Neural Networks"
  https://distill.pub/2021/gnn-intro
- G*eometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*
  https://geometricdeeplearning.com
- *Graph Neural Networks: Foundations, Frontiers, and Applications*
  https://graph-neural-networks.github.io

However, I think more beginner-friendly, from-zero-to-hero expositions of mathematics of the field are yet to be proposed. I hope to do this in the future, in continuation of this book. Before this ambition turns into a full-blooded book, I will most likely take a few stabs at it over my Math and Data blog, https://mathanddata.com, to test out the perception of the material. Needless to say, the simplest way to stay up to date on my content here or on other platforms is to sign up for the newsletter on the blog and follow me on LinkedIn.

Bibliography

[1] Hervé Abdi and Lynne J. Williams. Principal component analysis. 2(4):433–459, 2010. ISSN 1939-0068. doi: 10.1002/wics.101. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.101.

[2] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer, 1st ed. 2018 edition edition, 2018. ISBN 978-3-319-94462-3.

[3] David Anderson, Dennis Sweeney, Thomas Williams, Jeffrey Camm, James Cochran, Michael Fry, and Jeffrey Ohlmann. *Statistics for Business & Economics*. South-Western College Pub, 14th edition edition, 2019. ISBN 978-1-337-90106-2. URL https://amzn.to/428oavA.

[4] Sheldon Axler. *Linear Algebra Done Right*. Undergraduate Texts in Mathematics. Springer International Publishing, Cham, 2014. ISBN 978-3-319-11079-0 978-3-319-11080-6. doi: 10.1007/978-3-319-11080-6. URL https://amzn.to/3KO9gEY.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 1st ed. 2006. corr. 2nd printing 2011 edition edition, 2006. ISBN 978-0-387-31073-2. URL https://amzn.to/444SBVm.

[6] Joseph K. Blitzstein and Jessica Hwang. *Introduction to Probability*. Chapman and Hall/CRC, 1st edition edition, 2014. ISBN 978-1-4665-7557-8. URL https://amzn.to/40NnHhi.

[7] Leo Breiman. Random forests. 45(1):5–32, 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

[8] William Briggs, Lyle Cochran, Bernard Gillett, and Eric Schulz. *Calculus: Early Transcendentals*. Pearson, 3rd edition edition, 2018. ISBN 978-0-13-476364-4. URL https://amzn.to/3zSAT9E.

[9] George Casella and Roger Berger. *Statistical Inference*. Duxbury Press, 2 edition edition, 2001. ISBN 978-0-534-24312-8. URL https://amzn.to/3oRT6lF.

[10] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, illustrated edition edition, 2000. ISBN 978-0-521-78019-3. URL https://amzn.to/3HdEW4y.

[11] Rick Durrett. *Probability: Theory and Examples*. Cambridge University Press, 4th edition edition, 2010. ISBN 978-0-521-76539-8. URL https://amzn.to/3ACGWj9.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL https://www.deeplearningbook.org/. http://www.deeplearningbook.org.

[13] Geoffrey Grimmett and David Stirzaker. *Probability and Random Processes*. Oxford University Press, 3rd edition edition, 2001. ISBN 978-0-19-857222-0. URL https://amzn.to/3o7vWaq.

[14] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3rd edition edition, 2011. ISBN 978-93-80931-91-3. URL https://amzn.to/3HiKr1T.

[15] Scott Hartshorn. *Hypothesis Testing: A Visual Introduction To Statistical Significance*. Independently published, 2017. ISBN 978-1-973181-46-0. URL https://amzn.to/428PCcy.

[16] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition*. Springer, 2nd ed. 2009, corr. 9th printing 2017 edition edition, 2009. ISBN 978-0-387-84857-0. URL https://amzn.to/3AsE48l.

[17] H. Hotelling. Analysis of a complex of statistical variables into principal components. 24:417–441, 1933. ISSN 1939-2176. doi: 10.1037/h0071325. Place: US Publisher: Warwick & York.

[18] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2nd ed. 2002 edition edition, 2002. ISBN 978-0-387-95442-4.

[19] David G. Kleinbaum and Mitchel Klein. *Logistic Regression: A Self-Learning Text*. Springer, 3rd ed. 2010 edition edition, 2010. ISBN 978-1-4419-1741-6. URL https://amzn.to/3V2imS1.

[20] Alexandre Kowalczyk. Support vector machines succinctly, 2017. URL https://www.syncfusion.com/succinctly-free-ebooks/support-vector-machines-succinctly.

[21] Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*. Springer/Sci-Tech/Trade, 1st ed. 2013, corr. 2nd printing 2018 edition edition, 2014. ISBN 978-1-4614-6848-6. URL https://amzn.to/41BDcdB.

[22] Stanley Lemeshow, Rodney X. Sturdivant, and David W. Hosmer Jr. *Applied Logistic Regression*. Wiley, 3rd edition edition, 2013. ISBN 978-0-470-58247-3. URL https://amzn.to/3n1lHnW.

[23] Andy Liaw and Matthew Wiener. Classification and regression by RandomForest. 23, 2001. URL https://cogns.northwestern.edu/cbmg/LiawAndWiener2002.pdf.

[24] Carl Meyer. *Matrix Analysis and Applied Linear Algebra Book with Solutions Manual*. SIAM: Society for Industrial and Applied Mathematics, har/cdr edition edition, 2001. ISBN 978-0-89871-454-8. URL https://amzn.to/3ZRMErC.

[25] Douglas C. Montgomery, Elizabeth A. Peck, and G. Geoffrey Vining. *Introduction to Linear Regression Analysis*. Wiley, 5th edition edition, 2012. ISBN 978-0-470-54281-1. URL https://amzn.to/3LiLmBU.

[26] Alexey Natekin and Alois Knoll. Gradient boosting machines, a tutorial. 7, 2013. ISSN 1662-5218. URL https://www.frontiersin.org/articles/10.3389/fnbot.2013.00021.

[27] Michael A. Nielsen. Neural networks and deep learning. 2015. URL http://neuralnetworksanddeeplearning.com. Publisher: Determination Press.

[28] Karl Pearson. On lines and planes of closest fit to systems of points in space. 2 (11):559–572, 1901. ISSN 1941-5982. doi: 10.1080/14786440109462720. URL https://doi.org/10.1080/14786440109462720. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/14786440109462720.

[29] David Freedman Robert Pisani Roger Purves. *Statistics*. Viva, 4th edition edition, 2008. ISBN 978-81-309-1587-6. URL https://amzn.to/3LiP8da.

[30] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition*. Packt Publishing, 3rd ed. edition edition, 2019. ISBN 978-1-78995-575-0. URL https://amzn.to/3N2OGCi.

[31] Steven Roman. *Advanced Linear Algebra*. Springer/Sci-Tech/Trade, 3rd edition edition, 2007. ISBN 978-0-387-72828-5. URL https://amzn.to/40YMPmd.

[32] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, 10th edition edition, 2009. ISBN 978-0-12-375686-2. URL https://amzn.to/3Kvm3Lb.

[33] Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications. 2(2):169–194, 1998. ISSN 13845810. doi: 10.1023/A:1009745219419. URL http://link.springer.com/10.1023/A:1009745219419.

[34] Bernhard Schölkopf, Alexander J. Smola, and Francis Bach. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2001. ISBN 978-0-262-19475-4. URL https://amzn.to/3HdEW4y.

[35] George A. F. Seber and Alan J. Lee. *Linear Regression Analysis*. Wiley, 2 edition edition, 2003. ISBN 978-0-471-41540-4. URL https://amzn.to/41S2Nif.

[36] Jonathon Shlens. A tutorial on principal component analysis, 2014. URL http://arxiv.org/abs/1404.1100.

[37] Daren S. Starnes, Dan Yates, and David S. Moore. *The Practice of Statistics*. W. H. Freeman, fourth edition edition, 2010. ISBN 978-1-4292-4559-3. URL https://amzn.to/3LDDnQ8.

[38] James Stewart. *Calculus: Early Transcendentals*. Brooks Cole, 8th edition edition, 2015. ISBN 978-1-285-74155-0. URL https://amzn.to/3UxdIv5.

[39] James Stewart. *Multivariable Calculus*. Brooks Cole, 8th edition edition, 2015. ISBN 978-1-305-26664-3. URL https://amzn.to/412h8sd.

[40] Gilbert Strang. *Linear Algebra and Its Applications, 4th Edition*. Cengage Learning, 4th edition edition, 2006. ISBN 978-0-03-010567-8. URL https://amzn.to/3mfwYAQ.

[41] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Pearson, 1st edition edition, 2005. ISBN 978-0-321-32136-7. URL https://amzn.to/3L8rwYD.

[42] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, 1st edition edition, 1997. ISBN 978-0-89871-361-9. URL https://amzn.to/3KqEq42.

[43] William F. Trench. *Introduction to Real Analysis*. Pearson, 1st edition edition, 2002. ISBN 978-0-13-045786-8. URL https://amzn.to/3MD9oc0.

[44] Mario Triola. *Elementary Statistics*. Pearson, 13th edition edition, 2017. ISBN 978-0-13-446245-5. URL https://amzn.to/3oL18fT.

[45] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023. ISBN 978-1-00-938943-3. URL https://d2l.ai/index.html.

No Bullshit Math for Data Science provides a comprehensive and deep introduction to a wide range of mathematical topics that play crucial role in data science, such as linear algebra, calculus, statistics and probability theory, as well as their applications in machine learning. The book has two parts. In part I, the core topics mentioned above are explored in detail, with examples and visualizations for better illustration.

In part II, the book turns into the applications of the concepts learned in the first part in machine learning. Several well-known ML algorithms are studied from a mathematical perspective. All algorithms have also been given Python implementations where it is shown explicitly how to use the algorithms and their parameters in this programming language.

This book is intended for all practitioners of data science, including juniors and newcomers to the field. It can be treated as a reference book for the mathematical foundations of data science.