Linear Algebra Project

Chinmay Sharma (2022113005)
Chanukya Charvulu (2022101120)
Berelly Vishwanth (2022102008)

June 2023

Contents

1 Introduction and Acknowledgement **3**

2 Project Formulation **3**

3 Mathematical Background **3**
3.1 Matrices as Linear System of Equations 3
3.1.1 Gaussian Elimination 4
3.2 Matrices as Linear Transformations 6
3.3 Orthogonalization, Inner Products and Norms 8
3.3.1 Some Basic Definitions 8
3.3.2 Gram-Schmidt Process 10
3.3.3 Norms 12
3.4 Eigenvalues, Eigenvectors, and Diagonalization 13
3.4.1 What are eigenvalues and eigenvectors? 13
3.5 Matrix Decompositions 16
3.5.1 Singular Value Decomposition 16
3.5.2 LU Decomposition 18
3.5.3 QR Decomposition 20
3.6 Statistics and the Covariance Matrix 21
3.7 Vector Analysis 24
3.7.1 Partial Derivatives 24
3.7.2 Gradient as the Direction of Steepest Ascent 24
3.7.3 Jacobian as the Best Local Linear Map Approximation . 24
3.8 Optimization Theory Basics 25
3.8.1 Cost Functions 25
3.8.2 Error Functions 25
3.8.3 Iterative Optimization 26

4 Overview of Machine Learning Terminology and Methods 26
4.1 PAC Model . . . 27
4.2 Unsupervised and Supervised Learning . . . 27
4.2.1 Unsupervised Learning . . . 27
4.2.2 Supervised Learning . . . 27
4.3 Empirical Risk Minimization (ERM) . . . 28
4.4 Structural Risk Minimization (SRM) . . . 28
4.5 Generalization vs. Prediction Loss . . . 29
4.6 Connectivist Approaches . . . 29
4.7 Philosophy behind Machine Learning . . . 29

5 Applications 30
5.1 Regression Analysis . . . 30
5.1.1 Introduction and Applications . . . 30
5.1.2 Linear Regression . . . 30
5.1.3 Logistic Regression . . . 31
5.1.4 Other Relevant Regressions . . . 31
5.1.5 Worked Example: Linear Regression with Gradient Descent 32
5.2 Principal Component Analysis . . . 33
5.2.1 Principal Component Analysis (PCA) in Machine Learning 33
5.2.2 Applications of Principal Component Analysis (PCA) in Machine Learning . . . 34
5.3 Latent Semantic Analysis . . . 36
5.3.1 LSA Process Overview and Applications . . . 36
5.4 Applications of LSA in Machine Learning . . . 37
5.5 Support Vector Machines (SVMs) . . . 38
5.5.1 Introduction to SVMs . . . 38
5.5.2 Hard and Soft Boundaries: Generalization vs Training Error Tradeoff . . . 39
5.5.3 Hyperplanes and Affine Subspaces . . . 40
5.5.4 Kernel Tricks for Non-linear Classification and the Dual Problem . . . 40
5.5.5 Worked Example: Text Categorization with SVMs . . . 42
5.6 Deep Learning and Neural Networks . . . 43
5.6.1 Introduction and Scope of Neural Networks . . . 43
5.6.2 Basic Terminology . . . 43
5.6.3 Example: Neural Network for Cat Classification . . . 44

6 Individual Work Contribution. 45

7 Bibliography 46

1 Introduction and Acknowledgement

This project work was undertaken under our Linear Algebra final project to study Linear Algebra and its applications in Machine Learning.

I would like to express my gratitude towards Prof. Chittaranjan Hens and the Teaching assistants whose inputs were essential to the creation of this project.

Project Topic:

Linear Algebra and its Applications to Machine Learning.

Team members

1. Chinmay Sharma (2022113005)
2. Berelly Vishwanath (2022102008)
3. Chanukya Charvulu

2 Project Formulation

After surveying the literature, it was found that there was a research gap as there did not exist a self-contained introduction motivating the mathematical background required for machine learning techniques and also giving intuition for the same. Hence this project aims to do the following:

1. Be a self-contained introduction to the linear algebra concepts required in many widely used machine learning techniques
2. Motivate the intuition behind these techniques
3. Introduce the applications of these techniques in various fields

3 Mathematical Background

In this section we give a high-level overview of the techniques required to motivate the applications of Linear Algebra to Machine Learning which will be mentioned later.

3.1 Matrices as Linear System of Equations

Matrices can be used to solve linear systems of equations. By using matrices, we can find the solution and determine the number of solutions for a given system using Cramer's rule and Gaussian elimination. This is a prerequisite for starting machine learning since it enables ML algorithms to run on a large number of datasets.

A system of linear equations can be represented by an augmented matrix, where each row represents one equation in the system and each column represents a variable or constant term. The augmented matrix for the system is shown in Figure 1.

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{bmatrix}$$

Considering a general system of linear equations in 2 variables:

$$a_{11}x + a_{12}y = b_{11}$$
$$a_{21}x + a_{22}y = b_{21}$$

we can represent the above system as:

$$AX = B$$

where A is the coefficient matrix, X is the variable matrix, and B is the constants matrix. Hence, we have:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad X = \begin{bmatrix} x \\ y \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix}$$

The augmented matrix that represents the entire system of linear equations is $[A|B]$.

In general, a system of m linear equations with n variables has:

- Coefficient matrix A of order $m \times n$
- Variable matrix X of order $n \times 1$
- Constant matrix B of order $m \times 1$

3.1.1 Gaussian Elimination

Gaussian elimination, also known as the "row reduction algorithm," is a method for solving linear systems of equations. This method has several uses:

1. Finding the solution of a system of linear equations
2. Finding the rank of the given matrix by transforming it into row-echelon form, where the rank is the number of non-zero rows
3. Finding the determinant of a square matrix from its row-echelon matrix
4. Finding the inverse of the given matrix if it is invertible

The chief use of this method lies in reducing the given matrix to a row reduced echelon form, which can then conveniently give us a lot of information about the matrix.

Definition 3.1. *A matrix is said to be in row reduced echelon form (RREF) if it satisfies the following conditions:*

1. *All rows containing only zeros are at the bottom.*
2. *In each non-zero row, the leftmost non-zero entry is equal to 1, which is called a leading 1.*
3. *The leading 1 in each row is the only non-zero entry in its column.*
4. *Above and below each leading 1, all entries are zero.*

To transform a matrix into row-reduced echelon form or upper triangular form, we need to perform a series of elementary row operations until the matrix is row-reduced. The three basic elementary row operations include:

1. Swapping two rows $R_i \leftrightarrow R_j$
2. Multiplying a row by a non-zero scalar $R_i \to c \cdot R_i$
3. Adding a multiple of one row to another row $R_i \to R_i + c \cdot R_j$

By applying these elementary row operations, the augmented matrix is row-reduced. When we convert the matrix back into a system of linear equations, most of the variables are eliminated, making it easier to solve the equations since many coefficients become zero. There is a lot of information that can be obtained by the RREF by utilizing theorems such as the one given below.

Theorem 3.1. (**Rouche-Capelli**) For a system of linear equations in n variables the following hold:

1. The system has a solution if and only if the rank of the coefficient matrix A is equal to the rank of the augmented matrix $[A|B]$.
2. If $n = \text{rank}(A)$, then the system of linear equations has a unique solution.
3. Otherwise, there are infinitely many solutions.

For example, consider the following system of linear equations:

$$\begin{aligned} x + y + z &= 2 \\ x + 2y + 3z &= 5 \\ 2x + 4y + 3z &= 11 \end{aligned}$$

The augmented matrix for the given system is:

$$\begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 2 & 3 & 5 \\ 2 & 4 & 3 & 11 \end{bmatrix}$$

By applying the sequence of elementary row operations, we get:

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

Hence, the equations will now become:

$$\begin{aligned} x - z &= -1 \\ y + 2z &= 3 \\ 0 &= 4 \end{aligned}$$

Clearly, the above equations have no solution for all x, y, z.

3.2 Matrices as Linear Transformations

On a high level, the basic objects studied in linear algebra are abstract vector spaces and the transformations between them. Here we give their definitions and intuition without going into too much detail.

Definition 3.2. *A vector space V over a field $\mathbb{F}$ is a set equipped with two operations: vector addition and scalar multiplication, satisfying the following properties:*

1. ***Closure under addition:*** *For any vectors $\mathbf{u}, \mathbf{v} \in V$, their sum $\mathbf{u} + \mathbf{v}$ is also in V.*
2. ***Commutativity of addition:*** *For any vectors $\mathbf{u}, \mathbf{v} \in V$, we have $\mathbf{u}+\mathbf{v} = \mathbf{v} + \mathbf{u}$.*
3. ***Associativity of addition:*** *For any vectors $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$, we have $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$.*
4. ***Additive identity:*** *There exists a vector $\mathbf{0} \in V$, called the zero vector, such that for any vector $\mathbf{v} \in V$, we have $\mathbf{v} + \mathbf{0} = \mathbf{v}$.*
5. ***Existence of additive inverses:*** *For every vector $\mathbf{v} \in V$, there exists a vector $-\mathbf{v} \in V$ such that $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$.*
6. ***Closure under scalar multiplication:*** *For any scalar $\alpha \in \mathbb{F}$ and vector $\mathbf{v} \in V$, their product $\alpha\mathbf{v}$ is also in V.*
7. ***Compatibility of scalar multiplication with field multiplication:*** *For any scalars $\alpha, \beta \in \mathbb{F}$ and vector $\mathbf{v} \in V$, we have $(\alpha\beta)\mathbf{v} = \alpha(\beta\mathbf{v})$.*
8. ***Distributivity of scalar multiplication with respect to vector addition:*** *For any scalars $\alpha \in \mathbb{F}$ and vectors $\mathbf{u}, \mathbf{v} \in V$, we have $\alpha(\mathbf{u} + \mathbf{v}) = \alpha\mathbf{u} + \alpha\mathbf{v}$.*

9. ***Distributivity of scalar multiplication with respect to field addition:*** *For any scalars* $\alpha, \beta \in \mathbb{F}$ *and vector* $\mathbf{v} \in V$*, we have* $(\alpha + \beta)\mathbf{v} = \alpha\mathbf{v} + \beta\mathbf{v}$.

10. ***Scalar multiplication identity:*** *For any vector* $\mathbf{v} \in V$*, we have* $1\mathbf{v} = \mathbf{v}$*, where* 1 *is the multiplicative identity in* $\mathbb{F}$.

Hence, loosely speaking, vector spaces are mathematical spaces in which the objects follow our ordinary notion of addition and scalar multiplication. The morphisms between vector spaces are linear maps, which for finite dimensional vector spaces can be represented in the form of matrices

Definition 3.3. *Let* V *and* W *be vector spaces over the same field* $\mathbb{F}$*. A mapping* $T : V \to W$ *is called a linear transformation if it satisfies the following properties:*

1. ***Additivity:*** *For any vectors* $\mathbf{u}, \mathbf{v} \in V$*, we have* $T(\mathbf{u} + \mathbf{v}) = T(\mathbf{u}) + T(\mathbf{v})$.

2. ***Homogeneity:*** *For any scalar* $\alpha \in \mathbb{F}$ *and vector* $\mathbf{v} \in V$*, we have* $T(\alpha\mathbf{v}) = \alpha T(\mathbf{v})$.

Linear Transformations are therefore functions from one vector space to another that respect linear combinations. They must always map zero to zero so they are visualized as rotations and scaling.

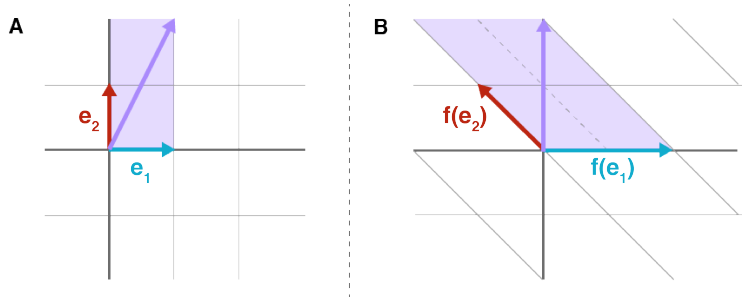

Figure 1: Visualization of a linear transformation

A key point to note here is that linear transformations are completely determined by where they map the basis vectors. This can be used to encode the information of a Linear transformation (for finite bases) into a matrix due to a fundamental result called the change of basis theorem

Theorem 3.2. Let V be a vector space of dimension n, and let $\mathcal{B} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n\}$ and $\mathcal{C} = \{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_n\}$ be two bases for V. If $[\mathbf{v}]_{\mathcal{B}}$ and $[\mathbf{v}]_{\mathcal{C}}$ denote the coordinate vectors of a vector $\mathbf{v}$ with respect to $\mathcal{B}$ and $\mathcal{C}$ respectively, then there

exists an invertible $n \times n$ matrix P such that:

$$[\mathbf{v}]_{\mathcal{C}} = P[\mathbf{v}]_{\mathcal{B}}$$

for all vectors $\mathbf{v}$ in V.

Proof. To prove the Change of Basis Theorem, we will explicitly compute the change of basis matrix P. Let $P = [\mathbf{w}_1 \ \mathbf{w}_2 \ \dots \ \mathbf{w}_n]$, where $\mathbf{w}_i$ is the i-th vector in $\mathcal{C}$.

Since $\mathcal{B}$ and $\mathcal{C}$ are bases for V, any vector $\mathbf{v}$ in V can be expressed as $\mathbf{v} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n$, where $c_1, c_2, \ldots, c_n$ are the coordinates of $\mathbf{v}$ with respect to $\mathcal{B}$.

We can rewrite this expression using the vectors in $\mathcal{C}$:

$$\mathbf{v} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \ldots + c_n\mathbf{v}_n = c_1(P^{-1}\mathbf{w}_1) + c_2(P^{-1}\mathbf{w}_2) + \ldots + c_n(P^{-1}\mathbf{w}_n)$$

Now, let $[\mathbf{v}]_{\mathcal{B}} = [c_1, c_2, \ldots, c_n]^\top$ be the coordinate vector of $\mathbf{v}$ with respect to $\mathcal{B}$. Using the expression above, we have:

$$[\mathbf{v}]_{\mathcal{C}} = [c_1(P^{-1}\mathbf{w}_1) + c_2(P^{-1}\mathbf{w}_2) + \ldots + c_n(P^{-1}\mathbf{w}_n)]_{\mathcal{C}}$$

$$[\mathbf{v}]_{\mathcal{C}} = [P^{-1}\mathbf{w}_1, P^{-1}\mathbf{w}_2, \ldots, P^{-1}\mathbf{w}_n] \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}$$

$$[\mathbf{v}]_{\mathcal{C}} = P \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = P[\mathbf{v}]_{\mathcal{B}}$$

Thus, we have shown that $[\mathbf{v}]_{\mathcal{C}} = P[\mathbf{v}]_{\mathcal{B}}$ for all vectors $\mathbf{v}$ in V. Therefore, the matrix P serves as the change of basis matrix, and it is invertible since $\mathcal{C}$ is a basis for V. □

There are other important theorems in linear algebra that are conducive to understanding matrices as linear maps such as the Rank-Nullity Theorem, which relates the rank and nullity of a linear transformation.

3.3 Orthogonalization, Inner Products and Norms

3.3.1 Some Basic Definitions

In linear algebra, several terms help us understand the properties and relationships between vectors. These terms include orthogonality, normalized vectors, and projection, which capture our intuition of vectors having "components" in the direction of another vector and the size of vectors.

Definition 3.4. ***Inner Product:*** *The inner product, also known as the dot product, is an operation that takes two vectors and produces a scalar value. Mathematically, for vectors* $\mathbf{v}$ *and* $\mathbf{w}$*, the inner product is denoted as* $\langle \mathbf{v}, \mathbf{w} \rangle$ *or* $\mathbf{v} \cdot \mathbf{w}$*. It is defined as the sum of the products of the corresponding components of the vectors. The inner product satisfies the following properties:*

1. ***Linearity in the First Argument:*** *For any vectors* $\mathbf{u}$*,* $\mathbf{v}$*, and* $\mathbf{w}$*, and any scalars* a *and* b*, we have:*

$$\langle a\mathbf{u} + b\mathbf{v}, \mathbf{w} \rangle = a\langle \mathbf{u}, \mathbf{w} \rangle + b\langle \mathbf{v}, \mathbf{w} \rangle$$

2. ***Symmetry:*** *For any vectors* $\mathbf{v}$ *and* $\mathbf{w}$*, we have:*

$$\langle \mathbf{v}, \mathbf{w} \rangle = \langle \mathbf{w}, \mathbf{v} \rangle$$

3. ***Positive Definiteness:*** *For any vector* $\mathbf{v}$*, we have:*

$$\langle \mathbf{v}, \mathbf{v} \rangle \geq 0$$

   The equality holds if and only if $\mathbf{v}$ *is the zero vector.*

These properties of the inner product allow us to quantify the notion of "similarity" or "closeness" between vectors and provide a foundation for various concepts in linear algebra and analysis.

Definition 3.5. ***Inner Product Space:*** *An inner product space is a vector space equipped with an inner product. It is a mathematical structure where vectors can be added together and multiplied by scalars, and an inner product is defined on the vectors in the space. The inner product space provides a framework for studying vector properties, orthogonality, and projections in a generalized setting.*

Definition 3.6. ***Orthogonality:*** *Two vectors are orthogonal if their inner product is zero. Mathematically, given two vectors* $\mathbf{v}$ *and* $\mathbf{w}$*, they are orthogonal if* $\langle \mathbf{v}, \mathbf{w} \rangle = 0$*. Geometrically, this means that the vectors are perpendicular to each other, and their directions do not overlap.*

Definition 3.7. ***Norm:*** *In an inner product space, the norm of a vector is a measure of its length or magnitude. Mathematically, for a vector* $\mathbf{v}$ *in an inner product space, the norm is denoted as* $\|\mathbf{v}\|$ *and is defined as the square root of the inner product of* $\mathbf{v}$ *with itself. It satisfies the following properties:*

1. ***Non-Negativity:*** *For any vector* $\mathbf{v}$*, the norm is non-negative:*

$$\|\mathbf{v}\| \geq 0$$

   The equality holds if and only if $\mathbf{v}$ *is the zero vector.*

2. ***Definiteness:*** *The norm is definite, meaning that it is zero if and only if the vector is the zero vector:*

$$\|\mathbf{v}\| = 0 \text{ if and only if } \mathbf{v} = \mathbf{0}$$

3. ***Homogeneity:*** *For any vector* $\mathbf{v}$ *and any scalar* a*, the norm satisfies:*

$$\|a\mathbf{v}\| = |a|\|\mathbf{v}\|$$

4. ***Triangle Inequality:*** *For any vectors* $\mathbf{u}$ *and* $\mathbf{v}$*, the norm satisfies:*

$$\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$$

The norm allows us to measure distances between vectors, define convergence in vector spaces, and quantify the magnitude or size of vectors in a mathematical framework. It plays a fundamental role in various branches of mathematics, including analysis, functional analysis, and numerical computations.

Definition 3.8. ***Normalized Vectors:*** *A vector is said to be normalized or unit-length if its magnitude or length is equal to 1. Mathematically, a vector* $\mathbf{v}$ *is normalized if* $\|\mathbf{v}\| = 1$*, where* $\|\mathbf{v}\|$ *denotes the Euclidean norm or length of* $\mathbf{v}$*. Normalized vectors are often used to represent directions in space and simplify calculations involving vectors.*

Definition 3.9. ***Projection:*** *The projection of a vector* $\mathbf{v}$ *onto another vector* $\mathbf{w}$ *represents the component of* $\mathbf{v}$ *in the direction of* $\mathbf{w}$*. It can be computed using the inner product and involves scaling the magnitude of* $\mathbf{v}$ *by the cosine of the angle between* $\mathbf{v}$ *and* $\mathbf{w}$*. The projection of* $\mathbf{v}$ *onto* $\mathbf{w}$ *is given by* $proj_{\mathbf{w}}(\mathbf{v}) = \frac{\langle \mathbf{v}, \mathbf{w} \rangle}{\|\mathbf{w}\|^2} \mathbf{w}$*.*

These linear algebra concepts play a fundamental role in various areas of mathematics and have practical applications in fields such as physics, computer graphics, signal processing, and machine learning. Understanding orthogonality, normalized vectors, and projection allows us to analyze vector relationships, decompose vectors into their components, and perform operations involving vectors in a geometric and algebraic framework.

3.3.2 Gram-Schmidt Process

We now come to a fundamentally important algorithm, which is called the Gram-Schmidt orthogonalization procedure. This algorithm makes it possible to construct, for each list of linearly independent vectors (resp. basis), a corresponding orthonormal list (resp. orthonormal basis).

Theorem 3.3. If $(v_1, \ldots, v_m)$ is a list of linearly independent vectors in V, then there exists an orthonormal list $(e_1, \ldots, e_m)$ such that

$$\text{span}(v_1, \ldots, v_k) = \text{span}(e_1, \ldots, e_k), \quad \text{for all } k = 1, \ldots, m.$$

Proof. The proof is **constructive**, that is, we will construct vectors $e_1, \ldots, e_m$ having the desired properties. Since $(v_1, \ldots, v_m)$ is linearly independent, $v_k \neq 0$ for each $k = 1, 2, \ldots, m$. Set $e_1 = \frac{v_1}{\|v_1\|}$. Then e_1 is a vector of norm 1 and satisfies Equation for $k = 1$. Next, set

$$e_2 = \frac{v_2 - \langle v_2, e_1 \rangle e_1}{\|v_2 - \langle v_2, e_1 \rangle e_1\|}.$$

This is, in fact, the normalized version of the orthogonal decomposition Equation. i.e.,

$w = v_2 - \langle v_2, e_1 \rangle e_1$,

where $w \perp e_1$. Note that $\|e_2\| = 1$ and $\text{span}(e_1, e_2) = \text{span}(v_1, v_2)$.

Now, suppose that $e_1, \ldots, e_{k-1}$ have been constructed such that $(e_1, \ldots, e_{k-1})$ is an orthonormal list and $\text{span}(v_1, \ldots, v_{k-1}) = \text{span}(e_1, \ldots, e_{k-1})$. Then define

$$e_k = \frac{v_k - \langle v_k, e_1 \rangle e_1 - \langle v_k, e_2 \rangle e_2 - \cdots - \langle v_k, e_{k-1} \rangle e_{k-1}}{\|v_k - \langle v_k, e_1 \rangle e_1 - \langle v_k, e_2 \rangle e_2 - \cdots - \langle v_k, e_{k-1} \rangle e_{k-1}\|}.$$

Since $(v_1, \ldots, v_k)$ is linearly independent, we know that $v_k \notin \text{span}(v_1, \ldots, v_{k-1})$. Hence, we also know that $v_k \notin \text{span}(e_1, \ldots, e_{k-1})$. It follows that the norm in the definition of e_k is not zero, and so e_k is well-defined (i.e., we are not dividing by zero). Note that a vector divided by its norm has norm 1 so that $\|e_k\| = 1$. Furthermore,

$$\begin{aligned}
\langle e_k, e_i \rangle &= \langle v_k - \langle v_k, e_1 \rangle e_1 - \langle v_k, e_2 \rangle e_2 - \cdots - \langle v_k, e_{k-1} \rangle e_{k-1}, e_i \rangle \\
&= \langle v_k, e_i \rangle - \langle v_k, e_i \rangle \frac{\|v_k - \langle v_k, e_1 \rangle e_1 - \langle v_k, e_2 \rangle e_2 - \cdots - \langle v_k, e_{k-1} \rangle e_{k-1}\|}{\|v_k - \langle v_k, e_1 \rangle e_1 - \langle v_k, e_2 \rangle e_2 - \cdots - \langle v_k, e_{k-1} \rangle e_{k-1}\|} \\
&= 0,
\end{aligned}$$

for each $1 \leq i < k$. Hence, $(e_1, \ldots, e_k)$ is orthonormal.

From the definition of e_k, we see that $v_k \in \text{span}(e_1, \ldots, e_k)$ so that $\text{span}(v_1, \ldots, v_k) \subset \text{span}(e_1, \ldots, e_k)$. Since both lists $(e_1, \ldots, e_k)$ and $(v_1, \ldots, v_k)$ are linearly independent, they must span subspaces of the same dimension and therefore are the same subspace. Hence Equation (9.5.1) holds.

□

Example: Take $v_1 = (1, 1, 0)$ and $v_2 = (2, 1, 1)$ in $\mathbb{R}^3$. The list (v_1, v_2) is linearly independent (as you should verify!). To illustrate the Gram-Schmidt procedure, we begin by setting

$$e_1 = \frac{v_1}{\|v_1\|} = \frac{1}{\sqrt{2}}(1, 1, 0).$$

Next, set

$$e_2 = \frac{v_2 - \langle v_2, e_1 \rangle e_1}{\|v_2 - \langle v_2, e_1 \rangle e_1\|}.$$

The inner product $\langle v_2, e_1 \rangle = \frac{1}{\sqrt{2}}\langle (1,1,0), (2,1,1) \rangle = \frac{3}{\sqrt{2}}$, so

$$u_2 = v_2 - \langle v_2, e_1 \rangle e_1 = (2,1,1) - \frac{3}{\sqrt{2}}(1,1,0) = \frac{1}{\sqrt{2}}(1,-1,2).$$

Calculating the norm of u_2, we obtain $\|u_2\| = \frac{1}{\sqrt{4}}\sqrt{1+1+4} = 1$, so

$$e_2 = \frac{u_2}{\|u_2\|} = \frac{1}{\sqrt{2}}(1,-1,2).$$

Therefore, (e_1, e_2) is an orthonormal list satisfying Equation (9.5.1). We can verify this by calculating the inner product:

$$\langle e_1, e_2 \rangle = \frac{1}{\sqrt{2}}\langle (1,1,0), (1,-1,2) \rangle = 0.$$

Hence, the Gram-Schmidt orthogonalization procedure has provided us with an orthonormal list (e_1, e_2) that spans the same subspace as the original list (v_1, v_2).

3.3.3 Norms

In linear algebra, norms are used to quantify the size or length of vectors or matrices. Three commonly used norms are the L1 norm, L2 norm, and Frobenius norm.

The L1 norm (also known as the Manhattan norm or the taxicab norm) of a vector $x = (x_1, x_2, \ldots, x_n)$ is defined as:

$$\|x\|_1 = |x_1| + |x_2| + \cdots + |x_n|$$

The L2 norm (also known as the Euclidean norm) of a vector x is defined as:

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

The Frobenius norm of a matrix A is defined as the square root of the sum of the squares of its elements:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}$$

The Frobenius norm is a measure of the "size" of a matrix and is commonly used in matrix analysis and optimization problems.

3.4 Eigenvalues, Eigenvectors, and Diagonalization

3.4.1 What are eigenvalues and eigenvectors?

- The prefix *eigen* is adopted from the German word for "innate" or "own".
- Eigenvectors of a square matrix are the nonzero vectors which, after being multiplied by the matrix, remain proportional to the original vector, i.e., they change only in magnitude, not in direction.
- For each eigenvector, the corresponding eigenvalue is the factor by which the eigenvector changes when multiplied by a matrix.
- Eigenvalues are also called latent values, and generally the letter λ (lambda) is used to represent eigenvalues.
- Let A be an $n \times n$ matrix, x be an $n \times 1$ column vector, and λ be a scalar. If $Ax = \lambda x$, then x is an eigenvector of A, and λ is an eigenvalue of A.
- Let A be an $n \times n$ matrix. We can find eigenvalues corresponding to it using the modification of the Cayley-Hamilton Theorem.

  Theorem 3.4. Cayley-Hamilton

  1. It states that if A is an $n \times n$ matrix, then the roots of the characteristic polynomial of the matrix are the eigenvalues of that matrix, and the corresponding eigenvectors can be found using the equation $Ax = \lambda x$.
  2. The characteristic polynomial of a matrix refers to the polynomial in λ obtained by the equation $|A - \lambda I| = 0$, i.e., the determinant of the matrix $A - \lambda I$ is zero, giving the polynomial in λ.
  3. Hence, we can say that the set of all eigenvectors corresponding to an eigenvalue λ of an $n \times n$ matrix A is just the set of nonzero vectors in the null space of $A - \lambda I$. It follows that this set of eigenvectors, together with the zero vector, is the eigen space of the eigenvalue λ and is denoted by $E(\lambda)$.

- We can also find the eigenvectors and eigenvalues of a given matrix geometrically by understanding the transformation done by the matrix to a vector geometrically. For example,

  Ex: Find the eigenvectors and eigenvalues of matrix A geometrically.

  $$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

  Sol: The given matrix A is definitely a reflection matrix in the x-axis. Any vector with coordinates (x, y) becomes $(x, -y)$ on multiplication with this matrix. Hence, the only vectors that map parallel to themselves after multiplication are vectors parallel to the x-axis (eigenvalue 1) and vectors parallel to the y-axis (eigenvalue -1).

- For an invertible matrix A, if λ is an eigenvalue, then $1/\lambda$ will be an eigenvalue of A^{-1}, but the corresponding eigenvectors are the same. Also, if λ is an eigenvalue of A, then λ is also an eigenvalue of A^T.
- Judging from the characteristic polynomial, we can say that
  1. The sum of eigenvalues of A is equal to the trace of A.
  2. The product of eigenvalues of A is the determinant of A.
- Geometric and Algebraic multiplicity of an eigenvalue:
  1. The geometric multiplicity of an eigenvalue λ of A is the dimension of its corresponding eigenspace $E_A(\lambda)$ or simply the dimension of the null space of $A - \lambda I$.
  2. The algebraic multiplicity of an eigenvalue λ of A is the number of times λ appears as a root in the characteristic polynomial of A, i.e., $P(A)$.
  3. In general, the algebraic and geometric multiplicities of an eigenvalue may differ, but the geometric multiplicity can never exceed the algebraic multiplicity.

Theorem 3.5 (Fundamental theorem of Invertible Matrices)**.** Let A be an $n \times n$ matrix. The following statements are equivalent:

1. A is invertible.
2. $Ax = b$ has a unique solution for every b in $\mathbb{R}^n$.
3. $Ax = 0$ has only the trivial solution.
4. The reduced row echelon form of A is I_n.
5. A is a product of elementary matrices.
6. Rank$(A) = n$.
7. Nullity$(A) = 0$.
8. Column vectors of A are linearly independent.
9. Column vectors of A form a basis for $\mathbb{R}^n$ and span $\mathbb{R}^n$.
10. Row vectors of A are linearly independent.
11. Row vectors of A form a basis for $\mathbb{R}^n$ and span $\mathbb{R}^n$.
12. $\det(A) \neq 0$.
13. 0 is not an eigenvalue of A.

Diagonalization: In many cases, the eigenvalue-eigenvector information contained within a matrix A can be displayed in a useful factorization of the form $A = PDP^{-1}$, where D is a diagonal matrix. This factorization enables us to quickly compute A^k for larger values of k, which is a fundamental idea in various fields of linear algebra and machine learning. Diagonalization refers to the transformation of a matrix into diagonal form.

1. An $n \times n$ matrix A is diagonalizable if and only if it has n linearly independent eigenvectors.
2. Specifically, $A = PDP^{-1}$ with D being a diagonal matrix if and only if the columns of P are n linearly independent eigenvectors of A, and the diagonal entries of D are the corresponding eigenvalues of A relative to the eigenvectors in P.
3. In other words, A is diagonalizable if and only if there are enough eigenvectors to form a basis of $\mathbb{R}^n$. This means A is diagonalizable if and only if the sum of the dimensions of the eigenspaces equals n.
4. An $n \times n$ matrix with n distinct eigenvalues is definitely diagonalizable.

Eigen Decomposition: The factorization of a square matrix A of order n with n linearly independent eigenvectors q_i and eigenvalues λ_i, where $i = 1, 2, \ldots, n$, into $A = QPQ^{-1}$ is also called the eigen decomposition of matrix A. Here, Q is a square $n \times n$ matrix whose ith column is the eigenvector q_i of A, and P is a diagonal matrix whose diagonal elements are the corresponding eigenvalues λ_i.

1. The n eigenvectors are usually normalized, but they need not be. A non-normalized set of n eigenvectors, v_i, can also be used as the columns of Q. The magnitude of the eigenvectors of Q gets canceled in the decomposition by the presence of Q^{-1}. If one of the eigenvalues λ_i has more than one linearly independent eigenvectors (that is, the geometric multiplicity of λ_i is greater than 1), then these eigenvectors for this eigenvalue λ_i can be chosen to be mutually orthogonal. However, when the eigenvectors belong to two different eigenvalues, it may not be possible for them to be orthogonal to each other.
2. The decomposition can be derived from the fundamental property of eigenvectors: $Av = \lambda v$.
3. **Matrix Inverse via Eigen Decomposition:**
   (a) If the matrix A can be eigen-decomposed as $A = QPQ^{-1}$ and if none of its eigenvalues are zero, then A is invertible and its inverse is given by $A^{-1} = QP^{-1}Q^{-1}$.
   (b) If A is a symmetric matrix, and since Q is formed by the eigenvectors of A, Q is guaranteed to be an orthogonal matrix, therefore $Q^{-1} = Q^T$. Also, because P is a diagonal matrix, its inverse is easy to calculate: $[A^{-1}]_{ii} = \frac{1}{\lambda_i}$.

Example: Diagonalize the given 2×2 matrix $A = \begin{bmatrix} \frac{1}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{1}{2} \end{bmatrix}$

Solution: First, calculate the characteristic polynomial to find the eigenvalues and eigenvectors. Hence, $f(\lambda) = \lambda^2 - \text{tr}(A)\lambda + \det(A) = \lambda^2 - \lambda - 2 = (\lambda + 1)(\lambda - 2)$. Thus, the eigenvalues are -1 and 2 for the given matrix. Since the above matrix of order 2 has 2 distinct eigenvalues, it is diagonalizable. Now, we have to calculate the eigenvectors for the corresponding eigenvalues. For $\lambda_1 = -1$, $(A + 1I)v = 0 \implies (A + I)v = 0$. On converting the above matrix into its reduced row echelon form, we get $v = 0$, hence the parametric form will be $x = -y$. Hence, the corresponding eigenvector for eigenvalue -1 will be $[-1\ 1]$. Similarly, the eigenvector corresponding to the eigenvalue 2 is $[1\ 1]$. Therefore, the eigenvectors v_1 and v_2 are linearly independent, and according to the diagonalization theorem, we can write $A = CDC^{-1}$ for $C = \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$ and $D = \begin{bmatrix} -1 & 0 \\ 0 & 2 \end{bmatrix}$, or if we choose $\lambda = 2$, we get $C = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $D = \begin{bmatrix} 2 & 0 \\ 0 & -1 \end{bmatrix}$.

3.5 Matrix Decompositions

3.5.1 Singular Value Decomposition

The Singular Value Decomposition is the first explicit tool that we will be utilizing for Machine Learning Applications. Here is the definition:

Definition 3.10. *Given an $m \times n$ matrix A, the Singular Value Decomposition (SVD) of A is a factorization of the form:*

$$A = U\Sigma V^T$$

where:

1. *U is an $m \times m$ orthogonal matrix, i.e., $UU^T = U^TU = I_m$,*
2. *V is an $n \times n$ orthogonal matrix, i.e., $VV^T = V^TV = I_n$,*
3. *Σ is an $m \times n$ matrix of the form:*

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_r \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

where $\sigma_1, \sigma_2, \ldots, \sigma_r$ are the non-zero singular values of A arranged in non-increasing order, and r is the rank of A.

The intuition behind the Singular Value Decomposition is basically that any transformation can be represented as some initial matrix (U) which represents the coordinates with respect to the original coordinate system, its scalings i.e. stretching or compressing($\sum$) in different directions and some rotation Matrix (V). If we see the $\vec{u_i}$ as $n \times 1$ column vectors, then the non-increasing order of singular values implies that the vectors have the corresponding importance in determining the matrix X (which will be later used in Principal Component Analysis).

Worked Example of SVD Consider the matrix $A = \begin{bmatrix} 5 & 5 \\ -1 & 7 \end{bmatrix}$.

The different matrices in an SVD are computed using the following relations:

$$A^T A = V\Sigma\Sigma^T V^T \tag{1}$$

$$AV = U\Sigma \tag{2}$$

To find the SVD of A, we start by computing $A^T A$:

$$A^T A = \begin{bmatrix} 5 & -1 \\ 5 & 7 \end{bmatrix} \begin{bmatrix} 5 & 5 \\ -1 & 7 \end{bmatrix} = \begin{bmatrix} 26 & 18 \\ 18 & 74 \end{bmatrix}$$

Next, we find the eigenvalues and eigenvectors of $A^T A$. The eigenvalues can be obtained by solving the characteristic equation $det|A^T A - \lambda I| = 0$:

$$|A^T A - \lambda I| = \begin{vmatrix} 26-\lambda & 18 \\ 18 & 74-\lambda \end{vmatrix} = (26-\lambda)(74-\lambda) - 18 \cdot 18 = \lambda^2 - 100\lambda + 40^2 = 0$$

Solving this equation, we find two eigenvalues: $\lambda_1 = 80$ and $\lambda_2 = 20$.

For each eigenvalue, we find the corresponding eigenvector by solving the system of equations $(A^T A - \lambda_i I)\mathbf{v}_i = \mathbf{0}$:

For $\lambda_1 = 80$, we have:

$$(A^T A - \lambda_1 I)\mathbf{v}_1 = \begin{bmatrix} 26-80 & 18 \\ 18 & 74-80 \end{bmatrix} \mathbf{v}_1 = \begin{bmatrix} -54 & 18 \\ 18 & -6 \end{bmatrix} \mathbf{v}_1 = \mathbf{0}$$

Solving this system of equations, we find that $\mathbf{v}_1 = \begin{bmatrix} 3/5 \\ 4/5 \end{bmatrix}$.

For $\lambda_2 = 20$, we have:

$$(A^T A - \lambda_2 I)\mathbf{v}_2 = \begin{bmatrix} 26-20 & 18 \\ 18 & 74-20 \end{bmatrix} \mathbf{v}_2 = \begin{bmatrix} 6 & 18 \\ 18 & 54 \end{bmatrix} \mathbf{v}_2 = \mathbf{0}$$

Solving this system of equations, we find that $\mathbf{v}_2 = \begin{bmatrix} -4/5 \\ 3/5 \end{bmatrix}$.

Next, we normalize the eigenvectors to obtain the orthonormal matrix V:

$$V = \begin{bmatrix} 3/5 & -4/5 \\ 4/5 & 3/5 \end{bmatrix}$$

Now, we compute Σ using the singular values, which are the square roots of the eigenvalues. This step in the computation also tells us why we have confined the values in Σ to non-negative values:

$$\Sigma = \begin{bmatrix} \sqrt{\lambda_1} & 0 \\ 0 & \sqrt{\lambda_2} \end{bmatrix} = \begin{bmatrix} \sqrt{80} & 0 \\ 0 & \sqrt{20} \end{bmatrix} = \begin{bmatrix} 4\sqrt{5} & 0 \\ 0 & 2\sqrt{5} \end{bmatrix}$$

Finally, we find the matrix U by computing $AV\Sigma^{-1}$:

$$U = AV\Sigma^{-1} = \begin{bmatrix} 5 & 5 \\ -1 & 7 \end{bmatrix} \begin{bmatrix} 3/5 & -4/5 \\ 4/5 & 3/5 \end{bmatrix} \begin{bmatrix} 1/(4\sqrt{5}) & 0 \\ 0 & 1/(2\sqrt{5}) \end{bmatrix} = \begin{bmatrix} 2/\sqrt{5} & -1/\sqrt{5} \\ 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix}$$

Therefore, the Singular Value Decomposition (SVD) of the matrix A is given by:

$$A = U\Sigma V^T = \begin{bmatrix} 2/\sqrt{5} & -1/\sqrt{5} \\ 1/\sqrt{5} & 2/\sqrt{5} \end{bmatrix} \begin{bmatrix} 4\sqrt{5} & 0 \\ 0 & 2\sqrt{5} \end{bmatrix} \begin{bmatrix} 3/5 & 4/5 \\ -4/5 & 3/5 \end{bmatrix}^T$$

3.5.2 LU Decomposition

Introduction: In linear algebra, LU Decomposition can be viewed as a matrix form of Gaussian elimination. Computers usually solve square systems of linear equations using LU Decomposition, and it is a key step in inverting a matrix or computing the determinant of a matrix. However, the limitation of this method is that it is limited to square matrices only.

Let A be a square matrix. An LU factorization refers to the factorization of A with proper row or column orderings or permutations into two factors: a unit lower triangular matrix L and an upper triangular matrix U. A lower unit triangular matrix means that all the entries of the main diagonal of L are 1. U need not be a unit upper triangular matrix. Mathematically, we have $A = LU$.

For example, if we consider A as a 2×2 matrix, then L and U are obtained as follows:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ can be factorized as } L = \begin{bmatrix} 1 & 0 \\ l_{21} & 1 \end{bmatrix} \text{ and } U = \begin{bmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{bmatrix}$$

To find the actual values of entries of L and U, given the matrix A, we can multiply the matrices L and U and solve the obtained system of linear equations. For example, for $A = \begin{bmatrix} 4 & 3 \\ 6 & 3 \end{bmatrix}$, we have the equations:

$$\begin{aligned} 1 \cdot u_{11} + 0 \cdot u_{12} &= 4 \\ 1 \cdot u_{11} + 1 \cdot u_{12} &= 3 \\ l_{21} \cdot u_{11} + 1 \cdot u_{12} &= 6 \\ l_{21} \cdot u_{11} + 1 \cdot u_{12} &= 3 \end{aligned}$$

Solving these equations, we get $u_{11} = 4$, $u_{12} = 3$, $l_{21} = 1.5$, and $u_{22} = -1.5$.

Types of LU Decomposition: There are different types of LU Decompositions in use for different applications:

1. LU Factorization with partial pivoting (LUP): It refers to LU factorization with row permutations only. We have $PA = LU$, where L and U are unit lower triangular and upper triangular matrices, and P is a permutation matrix that reorders the rows of A.

2. LU Factorization with Full pivoting: It involves both row and column permutations. We have $PAQ = LU$, where L, U, and P are defined as above, and Q is a permutation matrix that reorders the columns of A.

3. Lower-Diagonal-Upper Decomposition (LDU): We have $A = LDU$, where D is a diagonal matrix, L is a unit lower triangular matrix, and U is a unit upper triangular matrix.

Worked Example The process to find the solution to a system of linear equations using LU Decomposition is as follows:

Consider the following system of equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned}$$

To solve this system using LU decomposition:

1. Find the corresponding L and U for the given A using the method above or by converting A into row echelon form and manipulating it. We obtain L and U matrices.

2. Perform a forward substitution step by solving $LY = B$, where Y is a 3×1 vector. In our case, we have

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \text{ and } B = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

   Solving this system, we find $y_1 = 1$, $y_2 = 2$, and $y_3 = 5$.

3. Perform a backward substitution step by solving $UX = Y$, where X is the required solution vector. In our case, we have

$$X = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ and } Y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

   Solving this system, we find $x_1 = 1$, $x_2 = 0.5$, and $x_3 = -0.5$.

On analyzing the above process of solving systems of linear equations using LU Decomposition, we can say that the process becomes simpler when the coefficient matrix is either an upper or a lower triangular matrix, as it reduces the time and effort required to solve problems based on systems of linear equations.

3.5.3 QR Decomposition

The QR decomposition in linear algebra factorizes a matrix A into the product of an orthogonal matrix Q and an upper triangular matrix R, i.e., $A = QR$. This decomposition is useful in various applications, such as solving linear systems of equations and least squares problems.

Existence and Uniqueness: Given an $m \times n$ matrix A with linearly independent columns, the QR decomposition exists and is unique.

Procedure: To compute the QR decomposition of a matrix A, we can use the Gram-Schmidt process to orthogonalize the columns of A.

Let $a_1, a_2, \ldots, a_n$ be the columns of A. We start with $q_1 = \dfrac{a_1}{\|a_1\|}$, where $\|a_1\|$ denotes the Euclidean norm of a_1.

For $i = 2$ to n, we compute q_i as follows:

$$q_i = \frac{a_i - \sum_{j=1}^{i-1} \langle a_i, q_j \rangle q_j}{\|a_i - \sum_{j=1}^{i-1} \langle a_i, q_j \rangle q_j\|}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product.

After obtaining the orthogonal vectors $q_1, q_2, \ldots, q_n$, we can construct the matrix Q by placing these vectors as columns:

$$Q = \begin{bmatrix} q_1 & q_2 & \cdots & q_n \end{bmatrix}$$

The matrix R is then formed by multiplying $Q^\top A$:

$$R = Q^\top A$$

Note that R is an upper triangular matrix since the vectors q_i are orthogonal to each other.

Example Consider the matrix A given by:

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

To find the QR decomposition of A, we proceed as follows:

Step 1: Compute q_1.

$$q_1 = \frac{a_1}{\|a_1\|} = \frac{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}{\sqrt{6}}$$

Step 2: Compute q_2.

$$q_2 = \frac{a_2 - \langle a_2, q_1 \rangle q_1}{\|a_2 - \langle a_2, q_1 \rangle q_1\|} = \frac{\begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} - \frac{4}{\sqrt{6}} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}{\sqrt{6}} = \begin{bmatrix} -\frac{1}{3} \\ \frac{2}{3} \\ \frac{2}{3} \end{bmatrix}$$

Step 3: Construct Q.

$$Q = \begin{bmatrix} q_1 & q_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{6}} & -\frac{1}{3} \\ \frac{2}{\sqrt{6}} & \frac{2}{3} \\ \frac{1}{\sqrt{6}} & \frac{2}{3} \end{bmatrix}$$

Step 4: Compute R.

$$R = Q^\top A = \begin{bmatrix} \frac{1}{\sqrt{6}} & \frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \\ -\frac{1}{3} & \frac{2}{3} & \frac{2}{3} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} \sqrt{6} & \frac{5}{\sqrt{6}} \\ 0 & \frac{4}{3\sqrt{6}} \end{bmatrix}$$

Therefore, the QR decomposition of A is given by:

$$A = QR = \begin{bmatrix} \frac{1}{\sqrt{6}} & -\frac{1}{3} \\ \frac{2}{\sqrt{6}} & \frac{2}{3} \\ \frac{1}{\sqrt{6}} & \frac{2}{3} \end{bmatrix} \begin{bmatrix} \sqrt{6} & \frac{5}{\sqrt{6}} \\ 0 & \frac{4}{3\sqrt{6}} \end{bmatrix}$$

3.6 Statistics and the Covariance Matrix

Here we aim to give a brief overview of some relevant terms from probability and statistics and then give intuition for the same, building up to the Covariance Matrix.

Definition 3.11 (Random Variable)**.** *A random variable X is a function that assigns a numerical value to each outcome in a probability space. It maps the sample space Ω to a set of possible values $X(\Omega)$ such that for any subset $B \subseteq X(\Omega)$, the event $\{\omega \in \Omega : X(\omega) \in B\}$ is measurable. Random variables can be either discrete or continuous, depending on whether their value set is countable or uncountable, respectively.*

Definition 3.12 (Probability Distribution)**.** *A probability distribution for a random variable X is a function that describes the probabilities of each possible value that X can take. For a discrete random variable, the probability distribution is given by the probability mass function (PMF), denoted as $P(X = x)$,*

which assigns a probability to each value $x \in X(\Omega)$. For a continuous random variable, the probability distribution is described by the probability density function (PDF), denoted as $f_X(x)$, where the probability of X falling within a specific interval is given by the integral of the PDF over that interval.

Though these definitions seem dense because they are rigorous, for most of our applications a Random Variable is nothing but a mapping of events (or sets of events) to real numbers. The probability density function can be thought of as a function that returns the numerical value associated with each of those sets of events when we input the set. This can be constructed by analyzing the mapping and the measure associated with events space which the Random Variable encodes.

Definition 3.13 (Mean of a Probability Distribution)**.** *The mean, or expected value, of a random variable X with a probability distribution is a measure of its central tendency. For a discrete random variable, the mean is defined as the weighted sum of the possible values, each weighted by its corresponding probability. It is denoted as μ_X or $E(X)$ and calculated as $\mu_X = \sum_{x \in X(\Omega)} x \cdot P(X = x)$. For a continuous random variable, the mean is given by the integral of the product of the variable's values and the PDF, i.e., $\mu_X = \int_{-\infty}^{\infty} x \cdot f_X(x)\, dx$.*

Definition 3.14 (Variance and Standard Deviation)**.** *The variance of a random variable X measures the spread or dispersion of its probability distribution. It quantifies the average squared deviation of X from its expected value, μ_X. The variance is denoted as $Var(X)$ and is defined as:*

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

Alternatively, for a discrete random variable, the variance can be calculated as:

$$Var(X) = \sum_{x \in X(\Omega)} (x - \mu_X)^2 \cdot P(X = x)$$

The square root of the variance, denoted as σ_X, is called the standard deviation of X and provides a measure of the average deviation of X from its mean.

The definition of the mean is just a formalization of our notions of averaging and is rather self-explanatory. The variance can be seen as a measure of 'how far' each data point is from the expected value of the distribution and together with the Standard Deviation it gives us information about how much a probability distribution is 'spread out' around the mean.

Definition 3.15 (Covariance of a Probability Distribution)**.** *The covariance between two random variables X and Y, with probability distribution functions $f_{XY}(x, y)$, measures their joint variability. It quantifies the degree to which X and Y change together. The covariance is denoted as $Cov(X, Y)$ and is defined as $Cov(X, Y) = \mathbb{E}[(X - \mu_X)(Y - \mu_Y)]$, where $\mathbb{E}[\cdot]$ represents the expectation operator. Alternatively, for discrete random variables, the covariance can be calculated as $Cov(X, Y) = \sum_{x,y}(x - \mu_X)(y - \mu_Y) \cdot P(X = x, Y = y)$.*

The covariance is by itself a mostly qualitative measure of how two random variables change with each other. If it is negative it indicates that an increase in x would decrease y i.e. they are changing in opposite directions. The numerical value of the covariance by itself cannot give us a relevant quantitative measure because it is dependent on the scale of the system. What is high covariance for one system might be low covariance for another one depending on the values of x and y. This is fixed by dividing by the standard deviation to obtain the **Correlation Coefficient**.

Definition 3.16 (Correlation Coefficient)**.** *The correlation coefficient measures the linear relationship between two random variables X and Y. It can accurately quantify the strength and direction of their linear association unlike Covariance. The correlation coefficient between X and Y is denoted as ρ_{XY} and is defined as:*

$$\rho_{XY} = \frac{Cov(X,Y)}{\sigma_X \sigma_Y}$$

where $Cov(X,Y)$ is the covariance between X and Y, σ_X is the standard deviation of X, and σ_Y is the standard deviation of Y.

The correlation coefficient takes values between -1 and 1, where:

1. *$\rho_{XY} = 1$ indicates a perfect positive linear relationship between X and Y.*
2. *$\rho_{XY} = -1$ indicates a perfect negative linear relationship between X and Y.*
3. *$\rho_{XY} = 0$ indicates no linear relationship between X and Y.*

Definition 3.17 (Random Vector)**.** *A random vector $\mathbf{X}$ is a collection of random variables defined on the same probability space. It can be represented as $\mathbf{X} = (X_1, X_2, \ldots, X_n)$, where $X_1, X_2, \ldots, X_n$ are individual random variables.*

The random vector basically represents multiple characteristics or measurements associated with a single outcome in the probability space.

Definition 3.18 (Covariance Matrix)**.** *The covariance matrix of a random vector $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ is a square matrix that represents the pairwise covariances between the components of $\mathbf{X}$. It is denoted as $\mathbf{Cov}(\mathbf{X})$ and is defined as:*

$$\mathbf{Cov}(\mathbf{X}) = \begin{bmatrix} Cov(X_1, X_1) & Cov(X_1, X_2) & \ldots & Cov(X_1, X_n) \\ Cov(X_2, X_1) & Cov(X_2, X_2) & \ldots & Cov(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(X_n, X_1) & Cov(X_n, X_2) & \ldots & Cov(X_n, X_n) \end{bmatrix}$$

The (i,j)-th element of the covariance matrix represents the covariance between X_i and X_j, and the diagonal elements represent the variances of the individual random variables.

3.7 Vector Analysis

3.7.1 Partial Derivatives

Partial derivatives are a fundamental concept in multivariable calculus that allows us to measure how a function changes with respect to each of its variables while keeping other variables constant. For a function $f(x_1, x_2, \ldots, x_n)$, the partial derivative with respect to a variable x_i is denoted as $\frac{\partial f}{\partial x_i}$.

The partial derivative $\frac{\partial f}{\partial x_i}$ represents the rate of change of f with respect to x_i alone. It measures how sensitive the function is to small changes in the variable x_i, while holding all other variables fixed. It is computed by differentiating the function with respect to x_i as if the other variables were constants.

Partial derivatives allow us to analyze how a function behaves along different dimensions of its input space. They provide valuable information about the function's local behavior, such as the direction and magnitude of its slope with respect to each variable.

3.7.2 Gradient as the Direction of Steepest Ascent

The gradient operator in multivariable calculus is a vector operator denoted by ∇. It is used to represent the direction of steepest ascent of a scalar function in a multidimensional space. The gradient operator is defined as the vector of partial derivatives of the function with respect to each variable. For a function $f(x_1, x_2, \ldots, x_n)$, the gradient is given by $\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}\right)$.

The directional derivative of a function measures the rate of change of the function in a specific direction. It tells us how the function changes as we move along a given vector direction. The directional derivative in the direction of a unit vector $\mathbf{v}$ is denoted by $D_{\mathbf{v}} f$ and is defined as the dot product of the gradient of f with $\mathbf{v}$: $D_{\mathbf{v}} f = \nabla f \cdot \mathbf{v}$. Geometrically, the directional derivative represents the slope of the tangent line to the function's graph in the direction of $\mathbf{v}$.

When the directional derivative is maximized, it corresponds to the direction of steepest ascent of the function. In other words, the gradient vector points in the direction of the greatest rate of increase of the function. The magnitude of the gradient represents the rate of change, while the direction of the gradient gives the direction of steepest ascent.

3.7.3 Jacobian as the Best Local Linear Map Approximation

The Jacobian matrix is a powerful tool in multivariable calculus that captures the local behavior of a function around a given point. It serves as the best linear approximation of a function near that point.

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$, the Jacobian matrix J is an $m \times n$ matrix whose entries are the partial derivatives of f with respect to the input variables. The Jacobian matrix is defined as:

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

The Jacobian matrix provides a linear approximation of the function f near a specific point. It represents the best linear transformation that approximates the local behavior of the function. The Jacobian matrix is useful for understanding how small changes in the input variables affect the output variables, as well as for studying properties like local linearity, differentiability, and invertibility of the function.

In applications, the Jacobian matrix plays a crucial role in optimization, numerical methods, physics, and other fields where the local behavior of functions needs to be analyzed and approximated.

3.8 Optimization Theory Basics

Optimization theory is a fundamental concept in mathematics and plays a crucial role in various fields, including linear algebra. In this section, we will explore the basics of optimization, including cost functions, error functions, and iterative optimization methods.

3.8.1 Cost Functions

In optimization, a cost function, also known as an objective function or loss function, measures the performance or quality of a system or model. It assigns a numerical value to different solutions or parameter settings.

Let's consider a function $f(x)$ that we want to optimize. The cost function associated with $f(x)$, denoted as $J(x)$, represents the value we want to minimize or maximize. In the case of minimization, the cost function is defined as:

$$J(x) = \text{minimize}\, f(x)$$

Similarly, in the case of maximization, the cost function is defined as:

$$J(x) = \text{maximize}\, f(x)$$

The choice of the cost function depends on the specific problem and application. Common examples include mean squared error (MSE), and cross-entropy.

3.8.2 Error Functions

Error functions are a specific type of cost function commonly used in regression and classification problems. They quantify the discrepancy between the predicted values and the actual values.

In regression problems, where we aim to predict continuous values, a popular error function is the mean squared error (MSE), defined as:

$$\text{MSE}(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where y represents the true values and $\hat{y}$ represents the predicted values.

In classification problems, where the goal is to assign instances to predefined categories, an error function commonly used is cross-entropy loss. For binary classification, the cross-entropy loss is defined as:

$$\text{Cross Entropy}(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where y represents the true labels and $\hat{y}$ represents the predicted probabilities.

3.8.3 Iterative Optimization

Iterative optimization methods are commonly employed to find the optimal solutions for optimization problems. These methods iteratively update the parameters or variables of interest to minimize or maximize the cost function.

One of the widely used iterative optimization algorithms is the gradient descent algorithm. It is used to minimize differentiable cost functions by updating the parameters in the direction of the negative gradient. The update rule is given by:

$$x_{k+1} = x_k - \alpha \nabla J(x_k)$$

where x_k is the parameter at iteration k, α is the learning rate, $\nabla J(x_k)$ is the gradient of the cost function with respect to x_k. The learning rate controls the step size in each iteration.

Other iterative optimization algorithms include Newton's method, stochastic gradient descent (SGD), and Adam optimizer, among others.

4 Overview of Machine Learning Terminology and Methods

In this section, we will provide an overview of several fundamental machine learning concepts and terms. We will discuss the PAC model, supervised vs Unsupervised Learning, Empirical Risk Minimization (ERM), Structural Risk Minimization (SRM), generalization vs. prediction loss, connectivist approaches, and the philosophy behind machine learning.

4.1 PAC Model

The Probably Approximately Correct (PAC) model is a theoretical framework that provides a foundation for understanding the learning capabilities of machine learning algorithms. It was introduced by Leslie Valiant in 1984. The PAC model focuses on learning from a finite set of examples and aims to ensure that a learning algorithm will produce a hypothesis that is "approximately correct" with high probability. The PAC model defines the trade-off between the sample size, the accuracy of the hypothesis, and the confidence level.

Let's denote the sample space as X and the target concept we want to learn as $c : X \to \{0, 1\}$. In the PAC model, we assume that there exists a distribution D over X, which generates the examples. Given a sample of m independent and identically distributed (i.i.d.) examples $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ drawn from D, the learning algorithm should output a hypothesis h with high probability such that the error of h, defined as the probability that $h(x) \neq c(x)$ for an unseen example x drawn from D, is bounded by some small value ε $(0 < \varepsilon < 1)$ with confidence $1 - \delta$ $(0 < \delta < 1)$.

4.2 Unsupervised and Supervised Learning

Machine learning algorithms can be broadly categorized into two main types: unsupervised learning and supervised learning. These two approaches differ in their objectives and the nature of the available training data.

4.2.1 Unsupervised Learning

In unsupervised learning, the algorithm aims to find patterns, structures, or relationships in the data without explicit guidance or labeled examples. It operates on unlabeled data, where there is no predefined target variable or ground truth to guide the learning process.

The goal of unsupervised learning is to discover inherent patterns or clusters in the data, uncover hidden structures, or reduce the dimensionality of the input space. Common techniques used in unsupervised learning include dimensionality reduction methods such as principal component analysis.

Unsupervised learning finds applications in various domains, including data exploration, anomaly detection, customer segmentation, and recommendation systems. By exploring the data without prior knowledge, unsupervised learning algorithms can provide valuable insights and reveal hidden patterns that can guide further analysis or decision-making processes.

4.2.2 Supervised Learning

In contrast to unsupervised learning, supervised learning relies on labeled training data, where each example is associated with a known target or output variable. The algorithm learns from this labeled data to make predictions or classify new, unseen instances.

Supervised learning aims to build a model that maps input features to their corresponding target variables. The model learns from the labeled data examples by inferring patterns, relationships, or decision boundaries between the input features and the target variables.

The training process involves minimizing an objective function or loss function that measures the discrepancy between the predicted outputs and the true labels. Common supervised learning algorithms include support vector machines (SVMs), logistic regression, and neural networks.

Supervised learning finds wide-ranging applications, including email spam filtering, sentiment analysis, image classification, medical diagnosis, and stock price prediction. By leveraging the labeled data, supervised learning algorithms can generalize from the training examples and make predictions or decisions on new, unseen data instances.

In conclusion, unsupervised learning and supervised learning are two fundamental approaches in machine learning. Unsupervised learning focuses on exploring and discovering patterns in unlabeled data, while supervised learning utilizes labeled data to learn the relationship between input features and target variables. Both approaches have their unique advantages and applications, and their choice depends on the specific problem and available data.

4.3 Empirical Risk Minimization (ERM)

Empirical Risk Minimization (ERM) is a principle that serves as the foundation for many machine learning algorithms. The goal of ERM is to find a hypothesis that minimizes the empirical risk, which is the average loss incurred by the hypothesis on the training examples.

Let's denote the hypothesis space as H, and the loss function as $\ell : H \times X \times \{0, 1\} \to \mathbb{R}^+$. Given a training dataset $T = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, the ERM principle seeks to find the hypothesis h_e that minimizes the empirical risk:

$$h_e = \arg\min_h \frac{1}{m} \sum_i \ell(h, x_i, y_i)$$

Common loss functions include the 0-1 loss, squared loss, hinge loss, and cross-entropy loss, depending on the nature of the learning task.

4.4 Structural Risk Minimization (SRM)

Structural Risk Minimization (SRM) is an approach that balances the trade-off between the empirical risk and the model complexity to prevent overfitting. Overfitting occurs when a hypothesis fits the training data too closely but fails to generalize well to unseen data.

To mitigate overfitting, SRM introduces a regularization term, often represented as $\Omega(h)$, that penalizes complex hypotheses. The goal is to find the hypothesis h_s that minimizes the regularized risk:

$$h_s = \arg\min_h \frac{1}{m} \sum_i \ell(h, x_i, y_i) + \lambda \Omega(h)$$

Here, $\lambda \geq 0$ is a hyperparameter that controls the trade-off between empirical risk and model complexity. The regularization term $\Omega(h)$ can take various forms, such as L1 or L2 regularization, which add the absolute or squared magnitudes of the parameters, respectively.

4.5 Generalization vs. Prediction Loss

In machine learning, generalization refers to the ability of a hypothesis to perform well on unseen data drawn from the same distribution. The generalization error is the expected difference between the true error of a hypothesis and its empirical error on the training data.

On the other hand, prediction loss refers to the error incurred by a hypothesis on a specific example or a set of examples. Prediction loss is typically used to evaluate the performance of a hypothesis on the test or validation data.

While minimizing the empirical risk aims to reduce the prediction loss on the training data, the ultimate goal is to minimize the generalization error, ensuring the hypothesis performs well on unseen data.

4.6 Connectivist Approaches

Connectivist approaches in machine learning draw inspiration from neural networks and the interconnectedness of nodes in the brain. These approaches focus on constructing models that learn through interconnected units or neurons, mimicking the behavior of biological neural networks.

Neural networks, such as deep learning models, employ layers of interconnected nodes (neurons) that process and transform input data to produce an output. Through training with large datasets and backpropagation algorithms, neural networks learn to extract features, discover patterns, and make predictions.

4.7 Philosophy behind Machine Learning

The philosophy behind machine learning centers on the idea that algorithms can learn patterns and make predictions by processing data, without explicitly programmed instructions. Instead of relying on manual rule-based systems, machine learning enables systems to automatically learn and adapt from data.

Machine learning leverages statistical and linear algebraic techniques to optimize model parameters and make informed decisions based on data. It enables tasks such as classification, regression, clustering, and reinforcement learning, with applications spanning various domains like computer vision, natural language processing, and robotics.

In conclusion, understanding the PAC model, ERM, SRM, generalization vs. prediction loss, connectivist approaches, and the philosophy behind machine

learning provides a solid foundation for exploring and implementing advanced machine learning algorithms and techniques. These concepts form the basis for building intelligent systems that learn from data and improve their performance over time.

5 Applications

5.1 Regression Analysis

Regression analysis is a widely used statistical technique in machine learning that aims to model the relationship between a dependent variable and one or more independent variables. It is commonly applied to solve various prediction and estimation problems. In this subsection, we will explore different types of regression models and their applications.

5.1.1 Introduction and Applications

Regression models are used in various fields, such as economics, finance, healthcare, and social sciences. They are particularly valuable in scenarios where we want to predict or estimate a continuous outcome based on one or more input variables. Some common applications include:

- Predicting housing prices based on features like location, size, and number of rooms.
- Estimating the impact of advertising expenditure on sales.
- Modeling the relationship between patient characteristics and disease progression.

5.1.2 Linear Regression

Linear regression is one of the most fundamental and widely used regression techniques. It assumes a linear relationship between the dependent variable and the independent variables. The goal is to find the best-fit line that minimizes the difference between the predicted values and the actual values.

In linear regression, the dependent variable is modeled as a linear combination of the independent variables. The model can be represented as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \epsilon$$

where Y is the dependent variable, $X_1, X_2, \ldots, X_n$ are the independent variables, $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients to be estimated, and ϵ represents the error term.

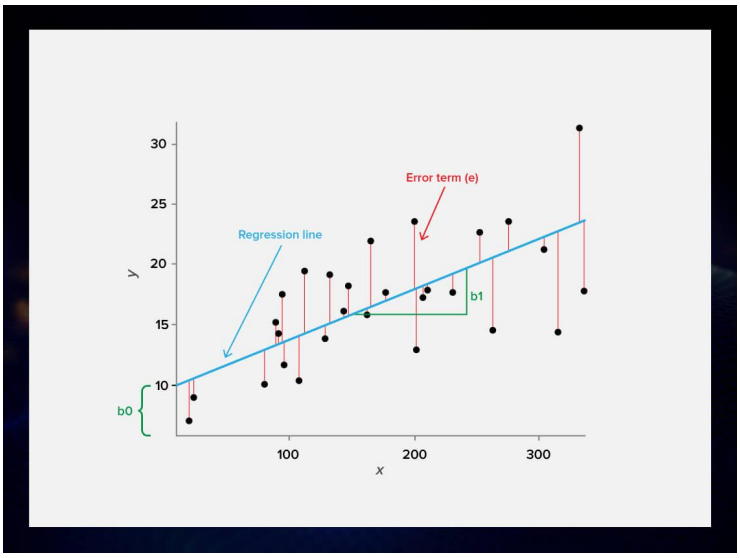

Figure 2: Linear Regression

5.1.3 Logistic Regression

Logistic regression is a type of regression used when the dependent variable is binary or categorical. It is commonly used for classification tasks where we want to predict the probability of an event or assign observations to different classes.

Unlike linear regression, logistic regression uses a logistic or sigmoid function to transform the linear combination of independent variables into a probability value between 0 and 1. The model can be expressed as:

$$P(Y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}}$$

where $P(Y = 1)$ is the probability of the positive class, $X_1, X_2, \ldots, X_n$ are the independent variables, and $\beta_0, \beta_1, \beta_2, \ldots, \beta_n$ are the coefficients to be estimated.

5.1.4 Other Relevant Regressions

Besides linear regression and logistic regression, there are various other regression techniques used in machine learning. Some notable ones include:

- Polynomial regression, which models the relationship between the dependent variable and the independent variables using higher-order polynomial functions.
- Ridge regression, which introduces a regularization term to the linear regression model to prevent overfitting.

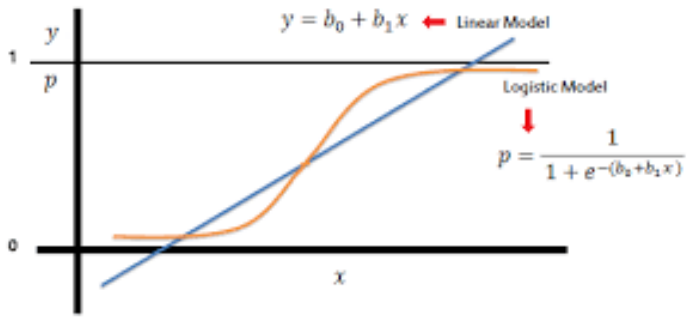

Figure 3: Logistic Regression

- Lasso regression, another regularization technique that encourages sparsity in the coefficient estimates by adding an L1 penalty term.
- Support Vector Regression (SVR), which uses support vector machines to perform regression tasks by finding a hyperplane that fits the data with a certain margin of tolerance.

5.1.5 Worked Example: Linear Regression with Gradient Descent

To illustrate the practical application of linear regression using gradient descent, let's consider a worked example of predicting the sales of a product based on advertising expenditure. We have collected data on advertising costs and corresponding sales for several periods, and we want to build a linear regression model to estimate sales based on advertising expenditure.

We start by formulating our linear regression model:

$$Y = \beta_0 + \beta_1 X$$

where Y represents the sales, X represents the advertising expenditure, β_0 is the intercept, and β_1 is the coefficient of the advertising expenditure.

To find the optimal values of β_0 and β_1, we employ the method of gradient descent. The objective is to minimize the sum of squared errors (SSE) between the predicted sales and the actual sales.

Using gradient descent, we update the coefficients iteratively as follows:

$$\beta_0^{(t+1)} = \beta_0^{(t)} - \alpha \frac{\partial}{\partial \beta_0} SSE$$
$$\beta_1^{(t+1)} = \beta_1^{(t)} - \alpha \frac{\partial}{\partial \beta_1} SSE$$

Here, α denotes the learning rate, and t represents the iteration step.

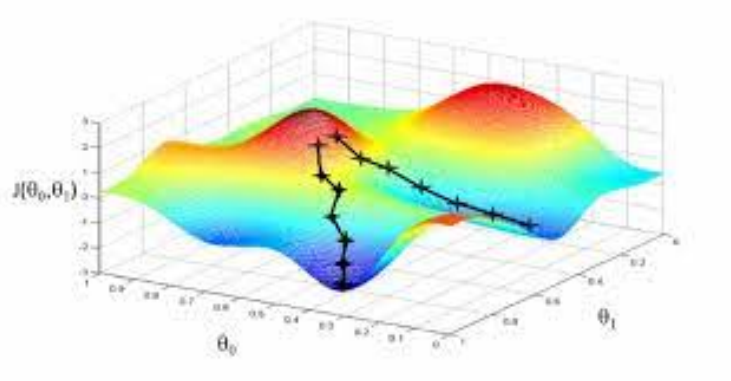

Figure 4: Linear Regression with Gradient Descent

To calculate the partial derivatives of SSE with respect to β_0 and β_1, we use the following formulas:

$$\frac{\partial}{\partial \beta_0} SSE = -2 \sum_{i=1}^{n} (Y_i - (\beta_0 + \beta_1 X_i))$$

$$\frac{\partial}{\partial \beta_1} SSE = -2 \sum_{i=1}^{n} X_i (Y_i - (\beta_0 + \beta_1 X_i))$$

Next, we perform the iterations until convergence or a predefined number of iterations. The final values of β_0 and β_1 obtained from the gradient descent process represent the best-fit line that minimizes the SSE.

With the optimal coefficients, we can utilize the linear regression model to make predictions for new advertising budgets by substituting the values of X.

This worked example demonstrates the practical implementation of linear regression using gradient descent. By applying this method, we can estimate sales based on advertising expenditure and build a predictive model for future sales projections.

5.2 Principal Component Analysis

5.2.1 Principal Component Analysis (PCA) in Machine Learning

PCA, a statistical technique widely used in machine learning, plays a crucial role in data visualization, feature extraction, data compression, and dimensionality reduction. It identifies patterns and relationships in high-dimensional datasets by transforming them into a lower-dimensional space.

The key concept behind PCA is to find principal components, which are linear combinations of the original variables. These components are ordered based on the amount of variation they explain in the data. While not a machine learning algorithm itself, PCA is frequently employed in machine learning pipelines

for data preprocessing, feature extraction, dimensionality reduction, and data visualization.

Linear algebra serves as the foundation for PCA, particularly in the following applications:

1. **Covariance matrix:** PCA computes the covariance matrix, representing relationships between variables. Computation involves linear algebra techniques such as matrix multiplication and eigenvalue decomposition.

2. **Eigenvalues and eigenvectors:** These play a crucial role in PCA. Eigenvalues and eigenvectors of the covariance matrix determine the principal components and quantify the variance explained by each component. Eigenvalue decomposition or singular value decomposition (SVD) are employed for obtaining these quantities.

3. **Projection:** PCA projects high-dimensional data onto a lower-dimensional subspace while preserving maximum variance. The linear transformation required for projection relies on linear algebra operations, including matrix multiplication.

4. **Dimensionality reduction:** By selecting a subset of eigenvectors corresponding to significant eigenvalues, PCA reduces data dimensionality. Linear algebra techniques like matrix multiplication and vector operations are used for this purpose.

5. **Reconstruction:** PCA allows the reconstruction of original data from a reduced-dimensional representation. Reconstruction involves projecting the reduced data back into the original high-dimensional space using matrix multiplication.

6. **Data visualization:** PCA facilitates the visualization of high-dimensional data in lower-dimensional spaces, enabling 2D or 3D plots. Linear algebra operations aid in mapping data points to the lower-dimensional space for visualization purposes.

By leveraging linear algebra, PCA enables various applications in machine learning, effectively reducing dimensionality while preserving essential information.

5.2.2 Applications of Principal Component Analysis (PCA) in Machine Learning

1. Data Visualization: PCA transforms high-dimensional data into a lower-dimensional space, enabling visualization in 2D or 3D plots. By reducing dimensionality, PCA facilitates the identification of clusters, trends, and outliers, aiding in the exploration and understanding of data structure and patterns.

2. Dimensionality Reduction and Noise Compression: PCA tackles the challenges of computational complexity and potential overfitting associated

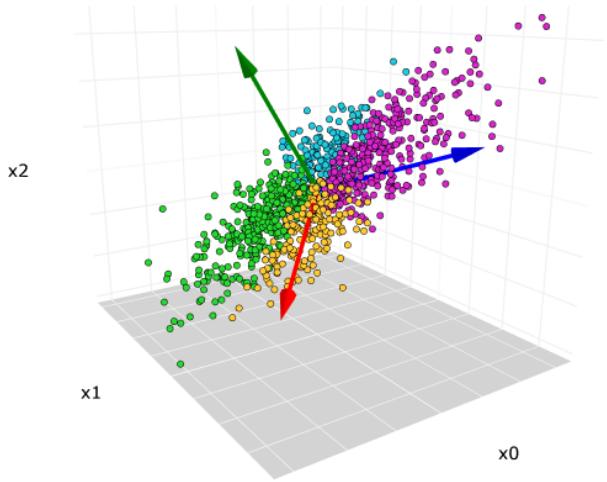

Figure 5: Principal Components Analysis

with high-dimensional datasets. By projecting data onto a lower-dimensional subspace defined by the principal components, PCA improves computational efficiency, mitigates the curse of dimensionality, and eliminates redundant or noisy features that could negatively impact the performance of machine learning algorithms.

3. Feature Extraction: PCA plays a crucial role in feature extraction by transforming high-dimensional data into a lower-dimensional space while preserving essential information. By selecting the most significant features or dimensions capturing the most substantial variations in the data, PCA aids in extracting relevant features for subsequent analysis.

4. Data Preprocessing: PCA is commonly employed as a preprocessing step before applying other machine learning algorithms. It helps remove redundant or noisy features, enhancing the performance of subsequent algorithms. Additionally, PCA assists in handling multicollinearity, a situation where predictors exhibit high correlation, which can lead to instability in models like linear regression.

5. Data Compression: By representing data using a smaller number of principal components, PCA reduces storage and memory requirements for the dataset. This reduction is particularly advantageous for large datasets or situations involving data transmission over networks with limited bandwidth.

These applications demonstrate the versatility and utility of PCA in various machine learning tasks, highlighting the significance of linear algebra operations such as matrix multiplication, eigenvalue decomposition, and vector operations in performing computations involved in PCA.

5.3 Latent Semantic Analysis

Latent Semantic Analysis (LSA) is a technique utilized in natural language processing and information retrieval to analyze the relationships between documents and terms within a large collection of text. It operates based on the notion that the meaning of words and documents can be inferred from their patterns of occurrence in a corpus, which is a collection of written texts.

LSA falls under the umbrella of unsupervised learning methods in NLP and is grounded on the distributional hypothesis, which assumes that words with similar meanings tend to occur in similar pieces of text. The LSA process involves several steps and finds applications in various domains:

5.3.1 LSA Process Overview and Applications

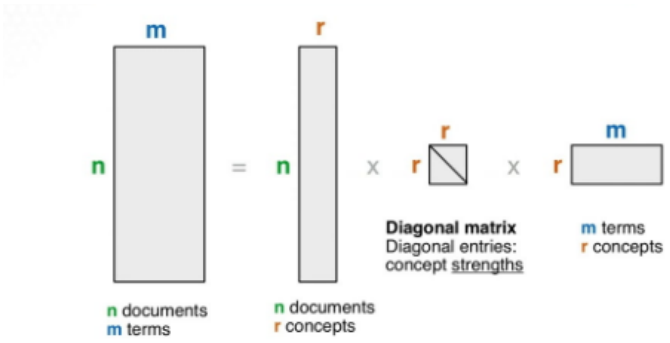

Figure 6: Latent Semantic Analysis

1. **Corpus Construction:** The text collection undergoes preprocessing, which involves removing stop words (e.g., "and," "the"), as well as stemming and lemmatization to reduce words to their base forms.

2. **Forming Term-Document Matrix:** A matrix is constructed where rows represent terms and columns represent documents. Each cell contains a numerical value representing the frequency or importance of the term in the document (e.g., term frequency-inverse document frequency, TF-IDF). The term-document matrix is then factorized using Singular Value Decomposition (SVD), which decomposes the matrix into three lower-rank matrices that capture the underlying semantic structure of the data.

3. **Singular Value Decomposition:** The term-document matrix is decomposed using SVD, resulting in three separate matrices: U, $\sum$, and V^T. Here, U represents the left singular vectors, $\sum$ is a diagonal matrix containing singular values, and V^T represents the right singular vectors.

4. **Dimensionality Reduction:** The resulting SVD matrices are truncated by retaining only the top k singular values or dimensions, effectively reducing the dimensionality of the data. This step helps remove noise and focus on the most significant relationships.

5. **Semantic Representation:** The reduced-dimensional representation of documents and terms can be used to calculate similarities between them. This enables tasks such as document clustering, document similarity ranking, information retrieval, and word sense disambiguation.

LSA plays a fundamental role in the overall LSA process, involving calculations such as singular value decomposition and the construction of the term-document matrix. However, LSA does have limitations, including its reliance on correlational statistics and its inability to capture complex semantic relationships, which have been addressed by newer advanced techniques.

5.4 Applications of LSA in Machine Learning

LSA finds extensive applications in the field of natural language processing (NLP), which is a subfield of machine learning. Some notable applications include:

1. **Information Retrieval:** LSA improves the effectiveness of search engines by capturing semantic relationships between documents and queries. By representing documents and queries as vectors in a high-dimensional space, LSA ranks queries based on semantic relevance, allowing retrieval of documents that share relevant concepts, even without exact keyword matches.

2. **Document Clustering:** LSA can group similar documents together based on their semantic content. By analyzing relationships between documents in the reduced-dimensional space, LSA uncovers latent topics and creates meaningful clusters, aiding in organizing large document collections.

3. **Question Answering Systems:** LSA assists question answering systems in identifying the most relevant documents or passages for a given query. By matching the semantic representation of questions with that of documents, LSA ranks and retrieves appropriate answers.

4. **Text Classification:** LSA aids in categorizing text documents into predefined classes or topics. By learning the relationships between documents and their semantic content, LSA provides features for machine learning algorithms to accurately classify new documents.

5. **Recommender Systems:** LSA can be employed in recommender systems to identify similar items or users based on their semantic profiles. By representing items or users in a lower-dimensional semantic space, LSA

suggests relevant items to users or finds similar users for collaborative filtering.

In summary, LSA creates a mathematical representation of the relationships between words and documents in a high-dimensional space. By leveraging SVD for matrix factorization, LSA reduces dimensionality and captures latent semantic relationships, resulting in a lower-dimensional representation that retains essential semantic information.

5.5 Support Vector Machines (SVMs)

Support Vector Machines (SVMs) are a powerful class of supervised machine learning algorithms used for classification and regression tasks. They have gained popularity due to their ability to handle high-dimensional data, robustness against overfitting, and versatility in handling linear and non-linear decision boundaries[1].

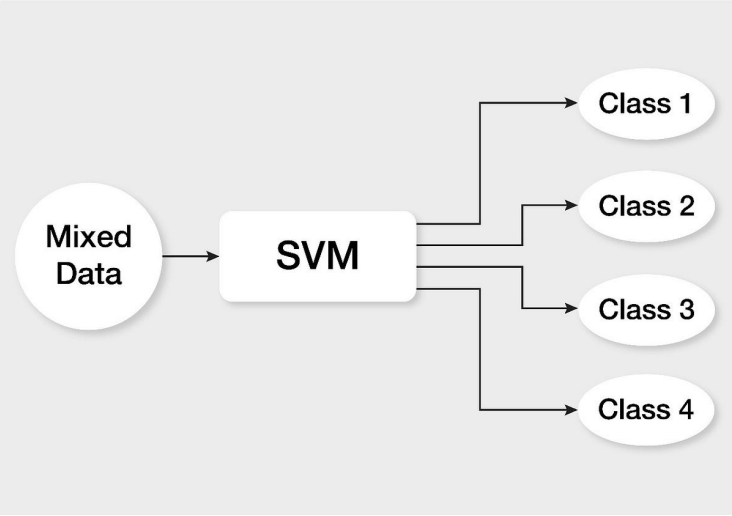

Figure 7: Support Vector Machines

5.5.1 Introduction to SVMs

SVMs are based on the concept of support vectors, which are the data points that lie closest to a decision boundary and play a crucial role in defining this optimal hyperplane which can help classify a set of points.

[1]S. Salcedo-Sanz, J. L. Rojo-Álvarez, M. Martínez-Ramón, G. Camps-Valls
Support vector machines in engineering: an overview

The hyperplane maximizes the margin, which is the distance between the hyperplane and the closest data points of each class. SVMs aim to find the hyperplane that achieves the maximum margin while minimizing the classification error.

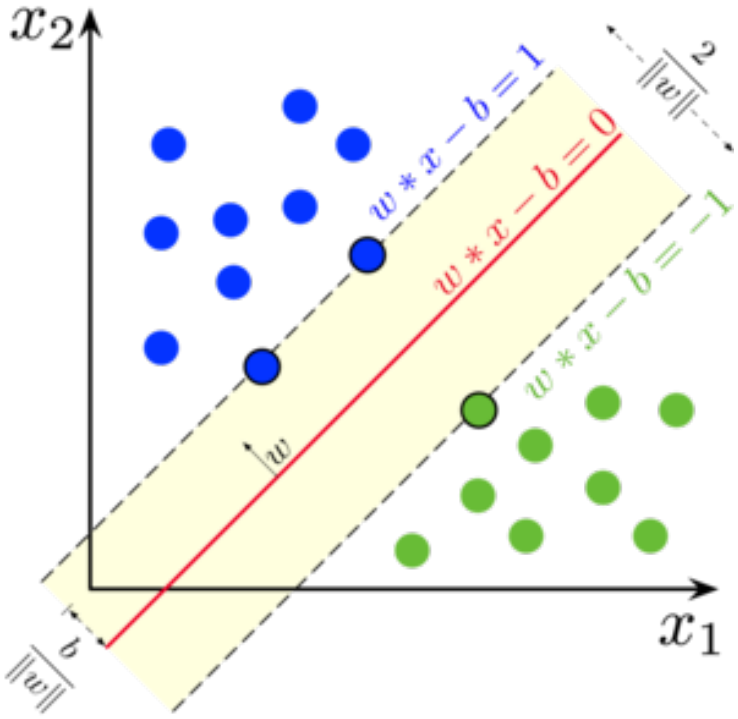

Figure 8: A support vector machine being used to find the optimal line (1D Hyperplane) which can be used to classify sets of points

SVMs have become popular in various domains, including computer vision, natural language processing, bioinformatics, and finance. Specific applications of SVMs include image classification, text categorization, gene expression analysis, and credit risk assessment.

5.5.2 Hard and Soft Boundaries: Generalization vs Training Error Tradeoff

In SVMs, the decision boundary is defined by a hyperplane that separates the data points into different classes. Hard boundaries refer to cases where the data can be perfectly separated by a hyperplane without any misclassifications. However, in real-world scenarios, it is often difficult or impossible to find a hard boundary due to noise or overlapping data.

Soft boundaries, on the other hand, allow for some misclassifications. They introduce a trade-off between maximizing the margin and allowing a certain

number of misclassified samples. The trade-off is controlled by a hyperparameter called the regularization parameter (C). A smaller value of C allows for more misclassifications, while a larger value of C enforces a stricter classification.

The choice of the regularization parameter (C) affects the trade-off between training error and generalization error. When C is set to a high value, the SVM aims to minimize the training error by classifying as many points correctly as possible. However, this may lead to overfitting, where the SVM becomes too specific to the training data and performs poorly on unseen data. On the other hand, when C is set to a low value, the SVM focuses more on achieving a larger margin and generalizing well to unseen data, but it may tolerate a higher training error.

The goal of SVMs is to strike a balance between minimizing the training error and optimizing the generalization error. By allowing a certain number of misclassifications with soft boundaries, SVMs can achieve better generalization performance and avoid overfitting. This ability to control the trade-off between training error and generalization error is one of the key strengths of SVMs.

5.5.3 Hyperplanes and Affine Subspaces

In SVMs, a hyperplane is a decision boundary that separates the data points into different classes. Mathematically, a hyperplane is defined by the equation:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

where $\mathbf{w}$ is the normal vector to the hyperplane and $\mathbf{x}$ is a data point. The sign of the expression $\mathbf{w} \cdot \mathbf{x} + b$ determines the class to which $\mathbf{x}$ belongs.

Affine subspaces, on the other hand, are lower-dimensional subspaces that can be tilted or translated relative to the origin. In SVMs, affine subspaces are used to define the decision boundaries in cases where the data is not linearly separable.

5.5.4 Kernel Tricks for Non-linear Classification and the Dual Problem

SVMs are inherently designed for linear classification. However, they can also handle non-linear data by using kernel functions. Kernel tricks enable SVMs to implicitly map the input data into a higher-dimensional feature space where linear separation is possible.

The key idea behind kernel tricks lies in expressing the SVM problem in terms of inner products between feature vectors. Let's denote the original input space as X and the corresponding feature space as F. The SVM objective can be reformulated using the concept of dual variables and the kernel function:

$$\text{maximize } \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

subject to

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \text{ for } i = 1, \ldots, n$$

where α_i are the dual variables, y_i is the target label for the i-th example, and $K(x_i, x_j)$ is the kernel function that calculates the inner product between the transformed feature vectors $\Phi(x_i)$ and $\Phi(x_j)$.

Commonly used kernel functions include the linear kernel, polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. The linear kernel ($K(x_i, x_j) = x_i^T x_j$) corresponds to no explicit transformation and operates in the original input space. The polynomial kernel ($K(x_i, x_j) = (x_i^T x_j + c)^d$) and the RBF kernel ($K(x_i, x_j) = \exp(-\gamma ||x_i - x_j||^2)$) implicitly map the input vectors to higher-dimensional feature spaces, enabling the SVM to capture non-linear decision boundaries. The sigmoid kernel ($K(x_i, x_j) = \tanh(\alpha x_i^T x_j + c)$) is also used to handle non-linear data.

By computing the kernel function, which effectively calculates the inner product between feature vectors in the transformed space, SVMs can classify non-linear data using linear separation in the feature space. This allows SVMs to capture complex decision boundaries and handle a wide range of non-linear classification tasks.

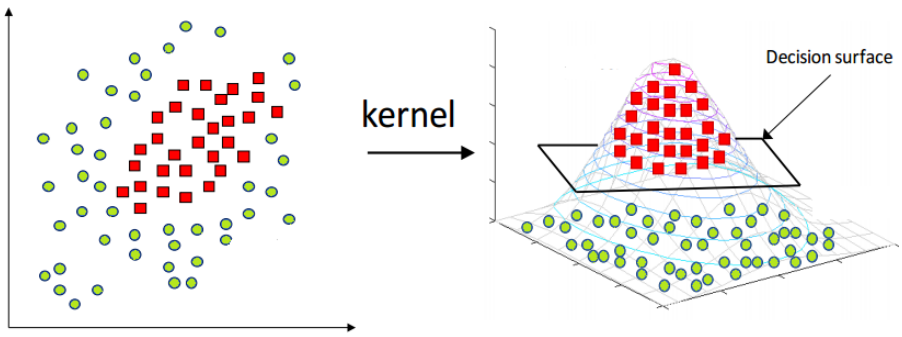

Figure 9: Kernel function being used to linearly classify a nonlinear boundary

The use of kernel tricks connects to the dual problem formulation of SVMs. By solving the dual problem, we can obtain the optimal dual variables α_i, which correspond to the support vectors in the training data. The dual solution involves only inner products between the support vectors, which can be efficiently computed using the kernel functions. This approach avoids the need to explicitly compute the transformation $\Phi(x)$, which can be computationally expensive or even infeasible in high-dimensional or infinite-dimensional feature spaces.

5.5.5 Worked Example: Text Categorization with SVMs

Let's consider an example of text categorization using SVMs. Suppose we have a dataset of news articles and want to classify them into different categories such as politics, sports, and entertainment.

We can represent each news article as a feature vector by applying techniques like TF-IDF (Term Frequency-Inverse Document Frequency) to convert the text into a numerical representation. TF-IDF calculates the importance of each term in the document based on its frequency within the document and its rarity across the entire collection. The resulting feature vector captures the relevance of different terms in the article. The SVM can then be trained on the labeled data using these feature vectors to learn the decision boundaries between different categories.

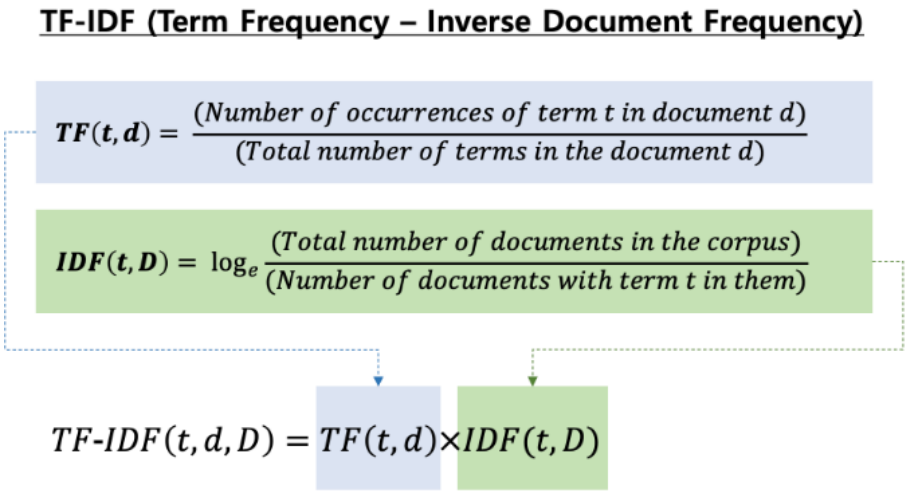

Figure 10: TF-IDF Technique used to represent vectorize news articles and classify

During the testing phase, new articles can be vectorized and classified based on the learned SVM model. SVMs have been shown to be effective in text categorization tasks due to their ability to handle high-dimensional data and capture complex decision boundaries.

This worked example demonstrates how SVMs can be applied to solve real-world problems, such as text categorization, by learning optimal decision boundaries from labeled data.

5.6 Deep Learning and Neural Networks

5.6.1 Introduction and Scope of Neural Networks

Neural networks are machine learning models inspired by the structure and functioning of biological brains. They consist of neurons which are nodes that take input, apply mathematical transformations, and produce outputs arranged in layers. They typically include an input layer, one or more hidden layers, and an output layer.

Operating on a connectivist paradigm, they can handle high-dimensional data and learn from it, making accurate predictions.

Leveraging linear algebra operations neural networks uncover hidden structures and excel in tasks such as classification, like accurately identifying objects in images when used in supervised learning settings. In unsupervised settings, they have applications such as generating new samples similar to the given data (GANs), dimensionality reduction, and clustering.

These networks also find applications in various fields for tasks like speech recognition, natural language processing, sentiment analysis, and recommendation systems. They have become significantly popular in recent years due to increased computational power, easier access to data, better algorithms and architectures etc.

5.6.2 Basic Terminology

The following terms are fundamental to understanding neural networks:

1. **Perceptrons:** Perceptrons are the basic building blocks of neural networks. They are artificial neurons that take input values, apply weights to them, and produce an output based on an activation function.

2. **Activation Function:** An activation function determines the output of a neuron given its weighted inputs. Common activation functions include the sigmoid function, the rectified linear unit (ReLU), and the hyperbolic tangent (tanh) function.

3. **Architecture:** The architecture of a neural network refers to its overall structure, including the arrangement and connectivity of its neurons. It defines the number of layers, the number of neurons in each layer, and the flow of information through the network.

4. **Layers:** Neural networks are organized into layers, with each layer containing a set of neurons. The input layer receives the input data, the output layer produces the final output, and the hidden layers are intermediate layers between the input and output layers.

5. **Multi-Layer Perceptrons (MLPs):** MLPs are neural networks composed of multiple layers of perceptrons. They are a fundamental type of feedforward neural network where information flows in one direction, from the input layer through the hidden layers to the output layer.

6. **Deep Learning:** Deep learning refers to the training and implementation of neural networks with multiple hidden layers. It aims to learn complex patterns

and representations from data by leveraging the depth and hierarchical structure of deep neural networks.

5.6.3 Example: Neural Network for Cat Classification

Let's consider a neural network model for classifying whether a given 64x64x3 vector corresponds to an image containing a cat or not.

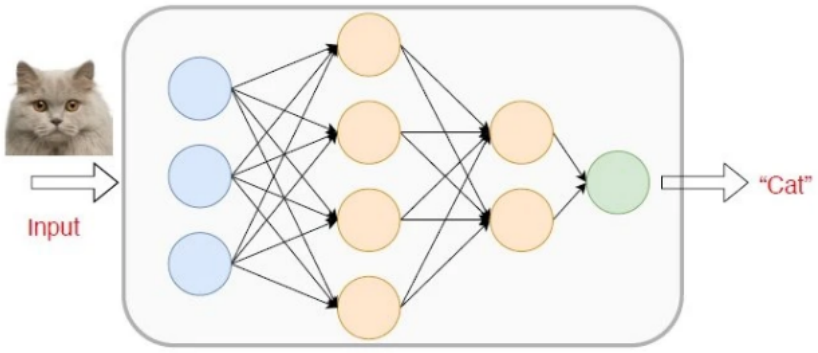

Figure 11: Neural Networks usage for image classification

We will denote the input vector as $\mathbf{x} \in \mathbb{R}^{64\times64\times3}$, and the output prediction as $\hat{y} \in \{0, 1\}$, where $\hat{y} = 1$ indicates the presence of a cat and $\hat{y} = 0$ indicates the absence of a cat.

The neural network consists of an input layer, one or more hidden layers, and an output layer. Let's assume a simple architecture with one hidden layer containing n_h neurons.

For simplicity, we'll denote the weights connecting the input layer to the hidden layer as $W^{[1]} \in \mathbb{R}^{n_h\times(64\times64\times3)}$, and the bias term as $b^{[1]} \in \mathbb{R}^{n_h\times1}$. Similarly, the weights connecting the hidden layer to the output layer are denoted as $W^{[2]} \in \mathbb{R}^{1\times n_h}$, and the bias term as $b^{[2]} \in \mathbb{R}$.

The forward propagation process of the neural network can be summarized as follows:

$$
\begin{aligned}
\mathbf{z}^{[1]} &= W^{[1]}\mathbf{x} + b^{[1]}, \\
\mathbf{a}^{[1]} &= g^{[1]}(\mathbf{z}^{[1]}), \\
z^{[2]} &= W^{[2]}\mathbf{a}^{[1]} + b^{[2]}, \\
\hat{y} &= g^{[2]}(z^{[2]}),
\end{aligned}
$$

where $g^{[1]}(\cdot)$ and $g^{[2]}(\cdot)$ are the activation functions applied to the hidden layer and output layer, respectively. In this case, we can use the sigmoid activation function:

$$g(z) = \frac{1}{1 + e^{-z}}.$$

During the training phase, the neural network learns the optimal values of the weight matrices $W^{[1]}$ and $W^{[2]}$, as well as the bias terms $b^{[1]}$ and $b^{[2]}$, by minimizing a suitable loss function such as the binary cross-entropy loss.

The complete training process involves iteratively updating the parameters using techniques like gradient descent or its variants.

Once trained, the neural network can make predictions on new images by propagating the input vector through the network and obtaining the output prediction $\hat{y}$.

This example demonstrates how a neural network can be mathematically formulated to classify whether cats exist in 64x64x3 vectors formed from flattening PNG images. The network's architecture, weight matrices, and activation functions play crucial roles in determining the accuracy of the classification results.

6 Individual Work Contribution.

1. Chinmay Sharma

   Contributions:-

   (a) **Mathematical Background-**Matrices as Linear Transformations
   (b) **Mathematical Background-**Basic Statistics and Covariance Matrix
   (c) **Mathematical Background-**Singular Value Decomposition
   (d) **Mathematical Background-**Basic Multivariable Calculus
   (e) **Applications-**Support Vector Machines
   (f) **Applications-**Gradient Descent and Linear/Logistic Regression
   (g) **Applications-**Deep Learning and Neural Networks

2. Chanukya Charvulu

   Contributions:-

   (a) **Mathematical Background-**Matrices as systems of linear Equations
   (b) **Mathematical Background-**Eigenvalues, Eigenvectors and Diagonalization
   (c) **Mathematical Background-**Lower Upper Decomposition
   (d) **Applications-**Principal Components Analysis
   (e) **Applications-**Latent Semantic Analysis

3. Berelly Vishwanth

Contributions:-

(a) **Mathematical Background-**Basic Optimization Theory

(b) **Mathematical Background-**Orthogonalization and Norms

(c) **Mathematical Background-**QR Decomposition

7 Bibliography

Here are the resources we have consulted, or have currently gathered after surveying the relevant state of the art literature from which inputs were taken for the project.

1. Machine learning Basics
2. Mathematical Background for Machine Learning
3. Logistic Regression
4. Support Vector Machines Research paper
5. Support Vector machines tutorial
6. Book motivating project mathematical background
7. neural networks book
8. Neural networks slides
9. Neural networks the hard way- tutorial website
10. Principal Components Analysis Research Paper 1
11. Principal Components Analysis Research Paper 2
12. Latent Semantic Analysis resource