Vectors

The Language of Data

A Data Structure Perspective

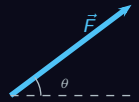
Dr Milan Joshi

Interactive PDF — Best viewed in Adobe Acrobat Reader

What is a Vector? — Three Perspectives

MODULE 1

Physicist



An arrow with magnitude and direction. A force, velocity, or displacement in space.

Computer Scientist

```
[58, 122, 85]
```

An ordered list of numbers. An array, a row in a dataset, a feature vector in ML.

Mathematician

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \in \mathbb{R}^n$$

A point in n-dimensional space. Each coordinate gives position along one axis.

Key Insight: All three views describe the *same object*. A vector is just a **structured list of numbers** — the interpretation depends on your domain.

Data Has Structure

A medical dataset:

Patient	Age	BP
Patient 1	58	122
Patient 2	71	110
Patient 3	48	110
Patient 4	34	123
Patient 5	62	152

$\Longrightarrow$ Each row is a **vector**! Formally:

$$\text{Patient}_1 = \begin{bmatrix} 58 \\ 122 \end{bmatrix} \qquad \text{Patient}_2 = \begin{bmatrix} 71 \\ 110 \end{bmatrix}$$

Definition (Boyd & Vandenberghe)

A **vector** is an **ordered finite list of numbers**. The **size** (dimension) is the number of elements. Each element is called a **component** or **entry**.

Data science view:

- Each **row** = one vector (data point)
- Each **column** = one feature/attribute
- **Dimension** = number of features

"A vector is a structured numerical fingerprint of data."

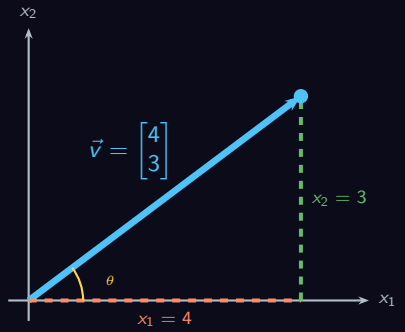
— A point in feature space, an arrow from the origin.

From Data to Vectors

Animated Visualization

Anatomy of a Vector

MODULE 2



Components:

- $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$ where each v_i is a component
- v_1 = horizontal extent
- v_2 = vertical extent

Dimension:

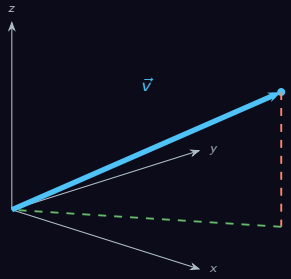
- Number of components = **dimension**
- $\vec{v} \in \mathbb{R}^2$ — a 2D vector
- $\vec{w} \in \mathbb{R}^n$ — an n-dimensional vector

Direction: $\theta = \arctan\left(\frac{v_2}{v_1}\right)$ **Magnitude:** $\|\vec{v}\| = \sqrt{v_1^2 + v_2^2}$

Vectors in Higher Dimensions

3D Vector:

$$\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \in \mathbb{R}^3$$



Beyond 3D — n-dimensional vectors:

- We **cannot visualize** $\mathbb{R}^n$ for $n > 3$
- But the **math works identically**
- In data science, n = **number of features**

Example: A patient with 5 measurements

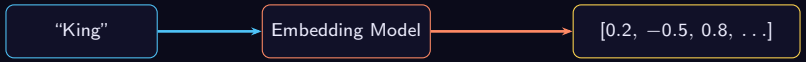
$$\vec{p} = \begin{bmatrix} 58 \\ 122 \\ 85 \\ 36.7 \\ 98 \end{bmatrix} \leftarrow \text{Age, BP, Weight, Temp, HR}$$

Each feature adds a dimension. Real ML models use vectors with **hundreds or thousands** of dimensions.

Vectors as Embeddings

MODULE 2

Text → Vector → Meaning



The magic of embeddings:

- Words, images, audio → dense vectors
- **Similar meanings → nearby vectors**
- Arithmetic on meaning!

$$\vec{\text{King}} - \vec{\text{Man}} + \vec{\text{Woman}} \approx \vec{\text{Queen}}$$

Roberts, LA for 21st Century

What gets embedded:

Data	Vector
Text / Words	Word2Vec, BERT
Images	CNN features, CLIP
Audio	Mel spectrograms
Graphs	Node2Vec

Modern AI = converting everything into vectors, then doing linear algebra on them.

Vectors Are Everywhere

MODULE 3

Real-World Examples

Physics: Force

$\vec{F} = \begin{bmatrix} F_x \\ F_y \end{bmatrix}$ Magnitude + Direction

Color: RGB

$\vec{c} = \begin{bmatrix} 255 \\ 165 \\ 0 \end{bmatrix}$ = Orange

Each pixel is a 3D vector!

Audio Signal

$\vec{s} = [s_1, s_2, \ldots, s_{44100}]$

1 second at 44.1kHz = 44100-dim vector

Portfolio

$\vec{w} = \begin{bmatrix} 100 \\ -50 \\ 200 \end{bmatrix}$ = Holdings Long/short positions in 3 stocks

Word Count (BoW)

"data science is fun" $\rightarrow$ [2, 2, 1, 1] counts per word

Customer Purchase

$\vec{p} = \begin{bmatrix} 100 \\ 50 \\ 75 \end{bmatrix}$ Spend per category

Pattern: Any structured collection of measurements can be represented as a vector.

Types of Vectors

Row vs Column Vector:

Row: $\vec{v}^T = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ $(1 \times n)$

Column: $\vec{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ $(n \times 1)$

Zero Vector:

$\vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ No magnitude, no direction

Ones Vector:

$\mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ Used for averaging: $\bar{x} = \frac{1}{n}\mathbf{1}^T\vec{x}$

Unit Vector:

$\|\hat{v}\| = 1$ Direction without magnitude

Standard basis: $\hat{\imath} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\hat{\jmath} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

Standard Basis Vectors:

$\vec{e}_i$ has 1 in position i, 0 elsewhere

$\vec{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \in \mathbb{R}^4$

Equal Vectors:

$\vec{a} = \vec{b}$ iff $a_i = b_i$ for all i

Same magnitude **and** same direction

Vectors in Python

MODULE 4

Code

```python
import numpy as np

# Column vector
v = np.array([58, 122])

# Zero vector
z = np.zeros(4)

# Ones vector
o = np.ones(4)

# Unit vector (normalize)
u = v / np.linalg.norm(v)

# Standard basis
e2 = np.eye(4)[1]
```

Key NumPy facts:

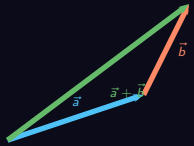
- `np.array` creates vectors
- `np.zeros(n)` $\rightarrow$ zero vector
- `np.ones(n)` $\rightarrow$ ones vector
- `np.eye(n)[i]` $\rightarrow \vec{e}_i$
- `v / np.linalg.norm(v)` $\rightarrow$ unit vector
- `v.shape` $\rightarrow$ dimension

Tip: NumPy 1D arrays are *neither* row nor column — they are just arrays. Use `v.reshape(-1,1)` for explicit column.

Vector Addition & Subtraction

Addition:

$$\vec{a} + \vec{b} = \begin{bmatrix} a_1 + b_1 \\ a_2 + b_2 \end{bmatrix}$$



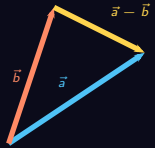
Triangle law: place $\vec{b}$ at tip of $\vec{a}$

Example:

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

Subtraction:

$$\vec{a} - \vec{b} = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \end{bmatrix}$$



$\vec{a} - \vec{b}$ points from tip of $\vec{b}$ to tip of $\vec{a}$

Properties:

Commutative: $\vec{a} + \vec{b} = \vec{b} + \vec{a}$
Associative: $(\vec{a} + \vec{b}) + \vec{c} = \vec{a} + (\vec{b} + \vec{c})$
Identity: $\vec{a} + \vec{0} = \vec{a}$
Inverse: $\vec{a} + (-\vec{a}) = \vec{0}$

Scalar Multiplication

MODULE 5

Definition:

$$k\vec{v} = \begin{bmatrix} kv_1 \\ kv_2 \\ \vdots \\ kv_n \end{bmatrix}$$

What happens:

$k > 1$	Stretches (same direction)
$0 < k < 1$	Shrinks (same direction)
$k = 0$	Zero vector $\vec{0}$
$k < 0$	Reverses direction
$k = -1$	Flips (additive inverse)

Example:

$$3 \cdot \begin{bmatrix} 2 \\ -1 \end{bmatrix} = \begin{bmatrix} 6 \\ -3 \end{bmatrix}$$

Distributive: $k(\vec{a} + \vec{b}) = k\vec{a} + k\vec{b}$

$(k + l)\vec{a} = k\vec{a} + l\vec{a}$

Norms — Measuring Vector Length

Euclidean Norm (L_2):

$$\|\vec{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Straight-line distance. Most common in ML.

Manhattan Norm (L_1):

$$\|\vec{v}\|_1 = \sum_{i=1}^{n} |v_i|$$

City-block distance. Robust to outliers.

Chebyshev Norm (L_∞):

$$\|\vec{v}\|_\infty = \max_i |v_i|$$

Maximum absolute component.

Example: $\vec{v} = \begin{bmatrix} 3 \\ -4 \end{bmatrix}$

$$\begin{aligned} \|\vec{v}\|_2 &= \sqrt{9+16} = 5 \\ \|\vec{v}\|_1 &= |3| + |-4| = 7 \\ \|\vec{v}\|_\infty &= \max(3,4) = 4 \end{aligned}$$

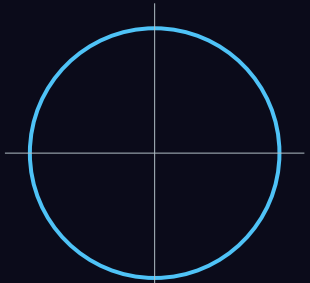
Properties of norms:

- $\|\vec{v}\| \geq 0$ (non-negative)
- $\|\vec{v}\| = 0 \iff \vec{v} = \vec{0}$
- $\|k\vec{v}\| = |k|\,\|\vec{v}\|$ (scaling)
- $\|\vec{a} + \vec{b}\| \leq \|\vec{a}\| + \|\vec{b}\|$ (triangle ineq.)

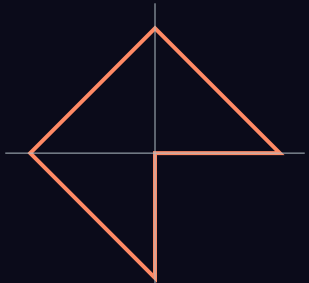
Geometry of Norms — Unit Balls

MODULE 6

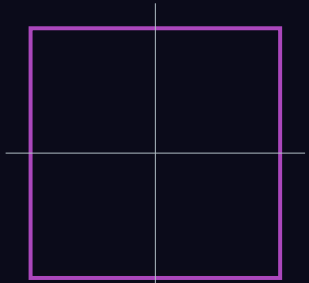
Visualization



L_2: Circle



L_1: Diamond



L_∞: Square

Unit ball = set of all vectors with $\|\vec{v}\| \leq 1$. The **shape** reveals how each norm "measures" distance. L_1 penalizes sparse solutions less $\Rightarrow$ used in **Lasso regularization**.

Distance Between Vectors

Definition:

$d(\vec{a}, \vec{b}) = \|\vec{a} - \vec{b}\|$

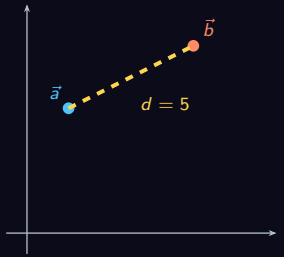
Euclidean distance:

$d_2(\vec{a}, \vec{b}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$

Example:

$\vec{a} = \begin{bmatrix}1\\3\end{bmatrix}$, $\vec{b} = \begin{bmatrix}4\\7\end{bmatrix}$

$d = \sqrt{(4-1)^2 + (7-3)^2} = \sqrt{9+16} = 5$



Python:

```
d = np.linalg.norm(a - b)
```

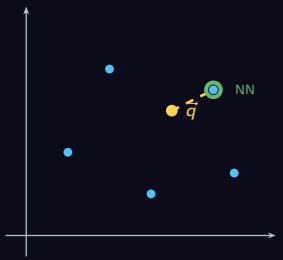
Small distance = similar data points.

This is the foundation of KNN, clustering, recommendation systems.

Nearest Neighbour Search

Problem: Given a query point $\vec{q}$, find the closest point in a dataset.

$$\text{NN}(\vec{q}) = \arg\min_{\vec{x}_i} \|\vec{q} - \vec{x}_i\|$$



Distance Matrix:

All pairwise distances:

$$D_{ij} = \|\vec{x}_i - \vec{x}_j\|$$

```
import numpy as np

X = np.array([[58,122],[71,110],

[48,110],[34,123]])

q = np.array([50, 115])

# Distances to all points

dists = [np.linalg.norm(q - x)

for x in X]

nn = np.argmin(dists)

print(f"Nearest:  {nn}")
```

Used in: KNN classification, recommendation engines, image search, anomaly detection.

The Dot Product

Algebraic definition:

$$\vec{a} \cdot \vec{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Geometric definition:

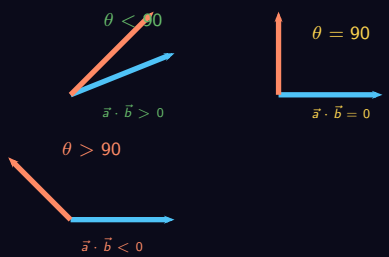
$$\vec{a} \cdot \vec{b} = \|\vec{a}\| \, \|\vec{b}\| \cos\theta$$

Example:

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \cdot \begin{bmatrix} 4 \\ -5 \\ 6 \end{bmatrix} = 4 + (-10) + 18 = 12$$

```
np.dot(a, b) # or a @ b
```

What the sign tells you:



Properties:

Commutative: $\vec{a} \cdot \vec{b} = \vec{b} \cdot \vec{a}$
Distributive: $\vec{a} \cdot (\vec{b} + \vec{c}) = \vec{a} \cdot \vec{b} + \vec{a} \cdot \vec{c}$

Self-dot: $\vec{a} \cdot \vec{a} = \|\vec{a}\|^2$

Dot Product Applications

MODULE 8

ML, Finance, NLP

ML: Weighted Score

Features $\vec{x}$, weights $\vec{w}$:

score $= \vec{w} \cdot \vec{x} = \sum w_i x_i$

This is a linear model's prediction!

Finance: NPV

Cash flows $\vec{c} = [c_0, c_1, \ldots, c_T]$

Discount $\vec{d} = [1, \frac{1}{1+r}, \ldots, \frac{1}{(1+r)^T}]$

NPV $= \vec{c} \cdot \vec{d}$

Cosine Similarity

$\cos\theta = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \, \|\vec{b}\|}$

$\cos\theta = 1$: identical direction

$\cos\theta = 0$: orthogonal

$\cos\theta = -1$: opposite

De-meaning

$\tilde{\vec{x}} = \vec{x} - \bar{x}\,\mathbf{1}$

Subtract mean from each component.

$\mathbf{1}^T \tilde{\vec{x}} = 0$ (zero sum)

The dot product is the most important single operation in all of machine learning.

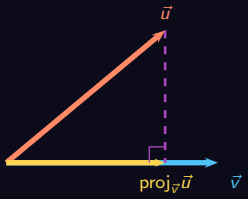
Projection of Vectors

Scalar projection:

$\text{comp}_{\vec{v}}\vec{u} = \frac{\vec{u}\cdot\vec{v}}{\|\vec{v}\|}$

Vector projection:

$\text{proj}_{\vec{v}}\vec{u} = \frac{\vec{u}\cdot\vec{v}}{\|\vec{v}\|^2}\,\vec{v}$



Example:

$\vec{u} = \begin{bmatrix}3\\4\end{bmatrix},\ \vec{v} = \begin{bmatrix}4\\3\end{bmatrix}$

$\vec{u}\cdot\vec{v} = 12 + 12 = 24$

$\|\vec{v}\|^2 = 16 + 9 = 25$

$\text{proj}_{\vec{v}}\vec{u} = \frac{24}{25}\begin{bmatrix}4\\3\end{bmatrix} = \begin{bmatrix}3.84\\2.88\end{bmatrix}$

```
proj = (np.dot(u,v)/np.dot(v,v))*v
```

Projection decomposes $\vec{u}$ into a component along $\vec{v}$ and a perpendicular residual. This is the geometric heart of **linear regression** and **PCA**.

Linear Combinations

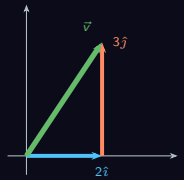
Definition:

$$\vec{v} = c_1\vec{v}_1 + c_2\vec{v}_2 + \cdots + c_k\vec{v}_k$$

where c_i are scalars (coefficients/weights)

Example:

$$2\begin{bmatrix}1\\0\end{bmatrix} + 3\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}2\\3\end{bmatrix}$$



Why it matters:

- **Span:** All possible linear combinations of a set of vectors
- **Basis:** Minimal spanning set
- **Linear independence:** No vector is a linear combination of the others

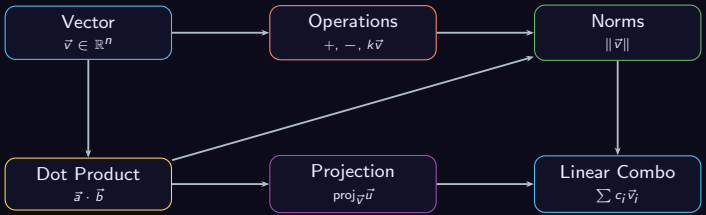
Applications:

- **Feature engineering**: new features as weighted sums
- **PCA**: data as combination of principal components
- **Linear regression**: $\hat{y} = X\vec{w}$ (prediction as lin. combo of features)
- **Neural networks**: each layer computes linear combinations

The Vector Toolkit — Summary

MODULE Σ

Everything Connects



The Big Picture: Vectors are the **universal language** of data. Every ML model, every neural network, every recommendation engine operates on vectors. Master these fundamentals, and the rest of linear algebra — matrices, transformations, eigenvalues — becomes a natural extension.

Next: Matrices, Linear Transformations, Eigenvalues & SVD