

Complete Vectors Tutorial: From Basics to Data Science & Machine Learning

Part 1: Vector Fundamentals

What is a Vector?

A vector is an ordered list of numbers. It represents both magnitude (length) and direction. In data science and ML, vectors are the basic building blocks for representing data.

Mathematical notation:

$$\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \text{or} \quad \mathbf{v} = [1, 2, 3] \text{ (horizontal notation)}$$

Visual representation (in 2D):

A 2D vector pointing to coordinates (2, 3)

Vector Dimensions

A vector with n elements is called an n -dimensional vector.

- **1D vector:** $[5]$ (single scalar)
- **2D vector:** $[3, 4]$ (point on a plane)
- **3D vector:** $[1, 2, 3]$ (point in 3D space)
- **100D vector:** $[x_1, x_2, \dots, x_{100}]$ (100 features)

In ML, we typically work with very high-dimensional vectors (hundreds to thousands of dimensions).

Vector Notation

Column vector ($m \times 1$): Row vector ($1 \times n$):

$\begin{bmatrix} x1 \\ x2 \\ x3 \\ \vdots \\ xm \end{bmatrix}$ $\begin{bmatrix} x1 & x2 & x3 & \dots & xn \end{bmatrix}$

$\begin{bmatrix} x1 \\ x2 \\ x3 \\ \vdots \\ xm \end{bmatrix}$

$\begin{bmatrix} x1 \\ x2 \\ x3 \\ \vdots \\ xm \end{bmatrix}$

$\begin{bmatrix} x1 \\ x2 \\ x3 \\ \vdots \\ xm \end{bmatrix}$

$\begin{bmatrix} x1 \\ x2 \\ x3 \\ \vdots \\ xm \end{bmatrix}$

Part 2: Vector Operations

1. Vector Addition

Add corresponding elements. Both vectors must have the same dimension.

Example (2D):

$$\begin{aligned} u &= \begin{bmatrix} 2 \\ 3 \end{bmatrix} & v &= \begin{bmatrix} 1 \\ 2 \end{bmatrix} & u + v &= \begin{bmatrix} 3 \\ 5 \end{bmatrix} \end{aligned}$$

Visual:

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

Example (3D):

$$\begin{aligned} u &= \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\ v &= \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} \\ u + v &= \begin{bmatrix} 1+4 \\ 2+5 \\ 3+6 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} \end{aligned}$$

Properties:

- $u + v = v + u$ (commutative)
- $u + (v + w) = (u + v) + w$ (associative)

- $u + 0 = u$ (adding zero vector)

2. Vector Subtraction

Subtract corresponding elements.

Example:

$$\begin{array}{ccc} u = [5] & v = [2] & u - v = [3] \\ [8] & [3] & [5] \end{array}$$

This is equivalent to $u + (-v)$, where $-v$ is the opposite direction.

3. Scalar Multiplication

Multiply each element by a single number (scalar).

Example:

$$\begin{array}{ccc} v = [2] & [2 \times 2] & [4] \\ [3] & 2 \times v = [2 \times 3] = [6] & \\ [1] & [2 \times 1] & [2] \end{array}$$

Geometric interpretation:

Original:	$2 \times$ scaled:
v	$2v$
	(twice as long)
*----	*-----

Properties:

- $\alpha(u + v) = \alpha u + \alpha v$ (distributive)
- $(\alpha + \beta)v = \alpha v + \beta v$
- $\alpha(\beta v) = (\alpha\beta)v$ (associative)
- $1v = v$
- $0v = 0$ (zero vector)

4. Dot Product (Inner Product)

Multiply corresponding elements and sum them. Results in a scalar.

Notation: $u \cdot v$ or $\langle u, v \rangle$

Formula:

$$u \cdot v = u_1 \times v_1 + u_2 \times v_2 + u_3 \times v_3 + \dots + u_n \times v_n$$

Example (2D):

$$u = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \quad v = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$u \cdot v = (3 \times 2) + (4 \times 1) = 6 + 4 = 10$$

Example (3D):

$$u = [1, 2, 3]$$

$$v = [4, 5, 6]$$

$$u \cdot v = (1 \times 4) + (2 \times 5) + (3 \times 6) = 4 + 10 + 18 = 32$$

Properties:

- $u \cdot v = v \cdot u$ (commutative)
- $u \cdot (v + w) = (u \cdot v) + (u \cdot w)$ (distributive)
- $(\alpha u) \cdot v = \alpha(u \cdot v)$
- $v \cdot v = \|v\|^2$ (related to magnitude)

5. Vector Magnitude (Norm)

The length or magnitude of a vector. Denoted as $\|v\|$ or $|v|$.

Euclidean norm (L2 norm):

$$\|v\| = \sqrt{v_1^2 + v_2^2 + v_3^2 + \dots + v_n^2}$$

Relationship to dot product:

$$\|v\| = \sqrt{v \cdot v}$$

Example (2D):

$$\mathbf{v} = [3, 4]$$

$$\|\mathbf{v}\| = \sqrt{(3^2 + 4^2)} = \sqrt{(9 + 16)} = \sqrt{25} = 5$$

Example (3D):

$$\mathbf{v} = [1, 2, 2]$$

$$\|\mathbf{v}\| = \sqrt{(1^2 + 2^2 + 2^2)} = \sqrt{(1 + 4 + 4)} = \sqrt{9} = 3$$

Other norms:

$$\text{L1 norm: } \|\mathbf{v}\|_1 = |\mathbf{v}_1| + |\mathbf{v}_2| + \dots + |\mathbf{v}_n|$$

$$\text{Example: } \mathbf{v} = [3, 4], \|\mathbf{v}\|_1 = 3 + 4 = 7$$

$$\text{L}\infty \text{ norm: } \|\mathbf{v}\|_\infty = \max(|\mathbf{v}_1|, |\mathbf{v}_2|, \dots, |\mathbf{v}_n|)$$

$$\text{Example: } \mathbf{v} = [3, 4, 1], \|\mathbf{v}\|_\infty = 4$$

6. Angle Between Vectors

The angle θ between two vectors can be found using the dot product.

Formula:

$$\cos(\theta) = (\mathbf{u} \cdot \mathbf{v}) / (\|\mathbf{u}\| \times \|\mathbf{v}\|)$$

$$\theta = \arccos((\mathbf{u} \cdot \mathbf{v}) / (\|\mathbf{u}\| \times \|\mathbf{v}\|))$$

Interpretation:

- $\theta = 0^\circ$: Vectors point in same direction (parallel)
- $\theta = 90^\circ$: Vectors are perpendicular (orthogonal)
- $\theta = 180^\circ$: Vectors point in opposite directions

Example:

$$\mathbf{u} = [1, 0] \quad \mathbf{v} = [1, 1]$$

$$\mathbf{u} \cdot \mathbf{v} = (1 \times 1) + (0 \times 1) = 1$$

$$\|\mathbf{u}\| = 1$$

$$\|\mathbf{v}\| = \sqrt{2}$$

$$\cos(\theta) = 1 / (1 \times \sqrt{2}) = 1/\sqrt{2} \approx 0.707$$

$$\theta = \arccos(0.707) \approx 45^\circ$$

7. Cross Product (3D only)

Produces a vector perpendicular to both input vectors.

Formula:

$$\mathbf{u} \times \mathbf{v} = [u_2 \times v_3 - u_3 \times v_2]$$

$$[u_3 \times v_1 - u_1 \times v_3]$$

$$[u_1 \times v_2 - u_2 \times v_1]$$

Example:

$$\mathbf{u} = [1, 0, 0]$$

$$\mathbf{v} = [0, 1, 0]$$

$$\mathbf{u} \times \mathbf{v} = [(0 \times 0 - 0 \times 1)] \quad [0]$$

$$[(0 \times 0 - 1 \times 0)] = [0]$$

$$[(1 \times 1 - 0 \times 0)] \quad [1]$$

Result: $[0, 0, 1]$ (perpendicular to both \mathbf{u} and \mathbf{v})

8. Vector Projection

Project one vector onto another.

Formula:

$$\text{proj}_{\mathbf{v}}(\mathbf{u}) = ((\mathbf{u} \cdot \mathbf{v}) / (\mathbf{v} \cdot \mathbf{v})) \times \mathbf{v}$$

Interpretation: Scalar version of how much \mathbf{u} goes in the direction of \mathbf{v} .

Example:

$$\mathbf{u} = [3, 4]$$

$$\mathbf{v} = [1, 0]$$

$$\mathbf{u} \cdot \mathbf{v} = 3$$

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

$$\text{proj}_{\mathbf{v}}(\mathbf{u}) = (3/1) \times [1, 0] = [3, 0]$$

This says \mathbf{u} extends 3 units in the direction of \mathbf{v}

Part 3: Special Vectors

Zero Vector

Vector with all zeros: $\mathbf{0} = [0, 0, 0, \dots]$

Properties:

- $\mathbf{0} + \mathbf{v} = \mathbf{v}$
- $\mathbf{0} \cdot \mathbf{v} = 0$
- $\|\mathbf{0}\| = 0$

Unit Vector

Vector with magnitude 1.

Common unit vectors in 3D:

$$\mathbf{e}_1 = [1, 0, 0]$$

$$\mathbf{e}_2 = [0, 1, 0]$$

$$\mathbf{e}_3 = [0, 0, 1]$$

$$\|\mathbf{e}_1\| = \|\mathbf{e}_2\| = \|\mathbf{e}_3\| = 1$$

Normalized Vector

Convert any vector to unit vector by dividing by its magnitude.

Formula:

$$u_{\text{normalized}} = u / \|u\|$$

Example:

$$v = [3, 4]$$

$$\|v\| = 5$$

$$v_{\text{normalized}} = [3/5, 4/5] = [0.6, 0.8]$$

$$\text{Check: } \|[0.6, 0.8]\| = \sqrt{(0.36 + 0.64)} = 1 \checkmark$$

Orthogonal Vectors

Two vectors are orthogonal if their dot product is zero: $u \cdot v = 0$

Example:

$$u = [1, 0, 0]$$

$$v = [0, 1, 0]$$

$$u \cdot v = 0 \rightarrow u \text{ and } v \text{ are orthogonal}$$

Orthonormal Vectors

Vectors that are orthogonal AND have unit magnitude.

Example:

$$v_1 = [1, 0, 0]$$

$$v_2 = [0, 1, 0]$$

$$v_3 = [0, 0, 1]$$

$$\text{All have magnitude 1 and } v_i \cdot v_j = 0 \text{ for } i \neq j$$

Part 4: Vector Spaces and Basis

Vector Space

A set of vectors where you can add vectors and multiply by scalars, and the results stay in the set.

Common vector spaces:

- \mathbb{R}^1 : All 1D vectors (real numbers)
- \mathbb{R}^2 : All 2D vectors (points on a plane)
- \mathbb{R}^3 : All 3D vectors
- \mathbb{R}^n : All n-dimensional vectors

Basis

A set of vectors that can represent all vectors in a space through linear combinations.

Requirements for basis:

1. Vectors must be linearly independent
2. Vectors must span the entire space

Example (2D):

Standard basis: $e_1 = [1, 0]$, $e_2 = [0, 1]$

Any 2D vector $[a, b] = a \times e_1 + b \times e_2$

Alternative basis: $v_1 = [1, 1]$, $v_2 = [1, -1]$

$[3, 1] = 2 \times [1, 1] + 1 \times [1, -1]$ ✓

Linear Independence

Vectors are linearly independent if no vector can be written as a combination of others.

Example (dependent vectors):

$u = [1, 2]$

$v = [2, 4] = 2 \times u$

These are linearly dependent

Example (independent vectors):

$$\mathbf{u} = [1, 0]$$

$$\mathbf{v} = [0, 1]$$

Neither can be written as a multiple of the other

They are linearly independent

Part 5: Applications in Machine Learning

1. Feature Vectors

Every data point is represented as a vector.

Example - Iris Dataset:

Flower sample:

- Sepal length: 5.1 cm
- Sepal width: 3.5 cm
- Petal length: 1.4 cm
- Petal width: 0.2 cm

Feature vector: $\mathbf{v} = [5.1, 3.5, 1.4, 0.2]$

This is a 4-dimensional vector representing one flower

Example - Image Data:

A 28×28 grayscale image (MNIST)

Flatten into a 1D vector of 784 pixels (28×28)

$\mathbf{v} = [\text{pixel1}, \text{pixel2}, \text{pixel3}, \dots, \text{pixel784}]$

Each pixel value is 0-255 (intensity)

Example - Text Data:

Document: "machine learning is great"

Word frequency vector:

$v = [\text{machine: } 1, \text{ learning: } 1, \text{ is: } 1, \text{ great: } 1, \dots]$

Or embeddings: $v = [0.2, -0.5, 0.1, 0.8, \dots]$ (word2vec)

2. Distance Metrics

Measure similarity between data points using vectors.

Euclidean Distance (L2):

$$d(u, v) = \|u - v\| = \sqrt{(u_1 - v_1)^2 + (u_2 - v_2)^2 + \dots + (u_n - v_n)^2}$$

Example:

$u = [1, 2]$

$v = [4, 6]$

$$d(u, v) = \sqrt{(1-4)^2 + (2-6)^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

Manhattan Distance (L1):

$$d(u, v) = \|u - v\|_1 = |u_1 - v_1| + |u_2 - v_2| + \dots + |u_n - v_n|$$

Example:

$u = [1, 2]$

$v = [4, 6]$

$$d(u, v) = |1-4| + |2-6| = 3 + 4 = 7$$

Cosine Similarity:

$$\text{similarity}(u, v) = (u \cdot v) / (\|u\| \times \|v\|)$$

Measures angle between vectors, not magnitude

Range: -1 to 1

Used in:

- k-NN clustering: Find nearest neighbors

- Anomaly detection: Points far from normal
- Recommendation systems: Similar users/items

3. Similarity and Distance in Text

Cosine Similarity Example:

Document 1: "cat and dog"

Vector: $d1 = [1, 1, 1, 0, 0]$ (cat, and, dog, bird, fish)

Document 2: "cat and bird"

Vector: $d2 = [1, 1, 0, 1, 0]$

$$d1 \cdot d2 = 1 + 1 + 0 = 2$$

$$\|d1\| = \sqrt{3}$$

$$\|d2\| = \sqrt{3}$$

$$\text{similarity} = 2 / (\sqrt{3} \times \sqrt{3}) = 2/3 \approx 0.67$$

More similar documents have cosine similarity closer to 1

4. Gradient Vectors

Direction of steepest increase in a function. Critical for optimization.

Example:

$$\text{Loss function: } L(w) = (w1 - 2)^2 + (w2 - 3)^2$$

$$\text{Gradient vector: } \nabla L = [2(w1 - 2), 2(w2 - 3)]$$

At point $w = [0, 0]$:

$$\nabla L = [2(0 - 2), 2(0 - 3)] = [-4, -6]$$

This vector points in direction of steepest increase

We move opposite: $w = w - \alpha \nabla L$ (gradient descent)

In Neural Networks:

Weights: w (vector of millions of parameters)

Loss: $L(w)$ (scalar)

Gradient: ∇L (vector with one value per parameter)

Update: $w = w - \text{learning_rate} \times \nabla L$

This uses vectors to update all weights simultaneously

5. Word Embeddings

Words represented as vectors in a continuous space.

Word2Vec / GloVe:

Word: "king" → Vector: $[2.5, -1.2, 0.8, \dots, 0.3]$ (300D)

Word: "queen" → Vector: $[2.3, -1.0, 0.9, \dots, 0.4]$

Word: "man" → Vector: $[1.2, -0.5, 0.6, \dots, 0.1]$

Word: "woman" → Vector: $[1.0, -0.3, 0.7, \dots, 0.2]$

Semantic relationships:

king - man + woman \approx queen (vectors capture meaning!)

BERT / Transformer Embeddings:

Input: "The cat sat on the mat"

Each word gets a 768-dimensional vector

Vectors capture contextual meaning

Similar words have vectors close together

6. Attention Mechanisms

Core concept in transformer models using vectors.

Scaled Dot-Product Attention:

Query vector: q (what we're looking for)

Key vector: k (what's available)

Value vector: v (what to retrieve)

Attention weight: $\alpha = \text{softmax}((q \cdot k) / \sqrt{d})$

Output: $\alpha \times v$ (weighted average)

This uses dot product to compute relevance

Example:

Processing: "The animal didn't cross the street because it was tired"

For word "it":

- Query: representation of "it"
- Keys: representations of all words
- Dot products: $q \cdot [\text{the, animal, didn't, ...}]$

Highest dot product likely with "animal"

Attention focuses on "animal" (it = animal)

7. Principal Component Analysis (PCA)

Reduce dimensions while preserving variance.

Process:

1. Represent data as vectors (samples \times features)
2. Compute covariance between feature vectors
3. Find eigenvectors (principal components) - these are vectors!
4. Project data onto these vectors

Each principal component is a vector

Projection: $v_{\text{projected}} = v \cdot \text{component1}, v \cdot \text{component2}, \dots$

Example:

Original: 1000 samples, 20 features (20D vectors)

PCA to 2 components

Get 2 principal component vectors: $pc1$, $pc2$

Project each sample:

$$x_{\text{new}} = [x \cdot pc1, x \cdot pc2]$$

Result: 1000 samples, 2 features (much easier to visualize!)

8. Clustering Algorithms

Vectors determine which cluster each point belongs to.

k-Means:

Initialize k cluster centers: $c1, c2, \dots, ck$ (vectors)

For each data point v :

1. Compute distance: $d(v, ci)$ for all clusters
2. Assign v to nearest cluster
3. Update cluster centers as average of assigned vectors

Repeat until convergence

Everything is vector operations!

Example:

Data points (2D vectors):

$$v1 = [1, 1]$$

$$v2 = [2, 2]$$

$$v3 = [10, 10]$$

$$v4 = [11, 11]$$

$k = 2$ clusters

Cluster 1: $\{v1, v2\}$ center: $[1.5, 1.5]$

Cluster 2: $\{v3, v4\}$ center: $[10.5, 10.5]$

Each point is a vector; centers are vectors

9. Recommendation Systems

Vector similarity finds similar users/items.

User-Item Vectors:

User 1 vector: [4, 5, 2, 3, 1] (ratings for 5 movies)

User 2 vector: [4, 5, 2, 3, 1] (nearly identical)

Cosine similarity ≈ 1.0 (very similar users)

→ Recommend movies User 2 liked to User 1

Embedding vectors capture latent features

Collaborative Filtering:

User vector: [preference for action, comedy, drama, ...]

Movie vector: [amount of action, comedy, drama, ...]

Prediction: score = user_vector · movie_vector

User who likes action (high value) × action movie (high value) = high score

10. Deep Learning Activations

Hidden layers produce vectors at each layer.

Neural Network Flow:

Input vector: x (n features)

↓ Matrix multiply × Weight matrix

Hidden 1: h_1 (m hidden units vector)

↓ Activation function (element-wise)

Hidden 2: h_2 (p hidden units vector)

↓ Matrix multiply × Weight matrix

Output: y (k classes vector)

Each hidden layer is a vector representation

These vectors learn increasingly abstract features

Example:

Image input: $x = [\text{pixel1}, \text{pixel2}, \dots, \text{pixel784}]$
Hidden layer 1: $h1 = [\text{edge1}, \text{edge2}, \dots, \text{edge128}]$ (edge patterns)
Hidden layer 2: $h2 = [\text{shape1}, \text{shape2}, \dots, \text{shape64}]$ (shapes)
Output: $y = [\text{prob_cat}, \text{prob_dog}, \text{prob_bird}]$ (classes)

Vectors transform from pixels \rightarrow edges \rightarrow shapes \rightarrow concepts

11. Transfer Learning

Pre-trained vectors capture useful information.

Example - Image Classification:

Pre-trained CNN (ImageNet):
Input: $[\text{pixel1}, \text{pixel2}, \dots, \text{pixel}224 \times 224]$
Output vector from layer before classification: [1000-D feature vector]

This 1000-D vector captures general image features
Use as input to a simple classifier for new task

The vector learned from 1.2M images transfers to new problem

Example - NLP:

BERT pre-trained on 3.3 billion words:
Input: "The cat sat"
Output: 3 vectors (one per word), 768-D each

These vectors understand English semantics
Use for downstream tasks:

- Sentiment analysis
- Named entity recognition
- Question answering

Transfer learning: Vectors from big data \rightarrow small data problems

12. Anomaly Detection

Unusual vectors stand out.

Isolation Forest:

Learn distribution of normal vectors

Anomaly = vector far from normal population

Measure using:

- Distance to nearest neighbors (euclidean)
- Density (local outlier factor)
- Reconstruction error (autoencoders)

Example: Network traffic vectors

Normal: [bytes_in: 1000, packets: 50, duration: 5]

Anomaly: [bytes_in: 1000000, packets: 500000, duration: 1]

Part 6: Vector Operations in Python

NumPy Basics

```
python
```

```
import numpy as np

# Create vectors
v1 = np.array([1, 2, 3])
v2 = np.array([4, 5, 6])

# Addition
v3 = v1 + v2 # [5, 7, 9]

# Subtraction
v3 = v1 - v2 # [-3, -3, -3]

# Scalar multiplication
v3 = 2 * v1 # [2, 4, 6]

# Dot product
dot = np.dot(v1, v2) #  $1 \times 4 + 2 \times 5 + 3 \times 6 = 32$ 

# Magnitude (norm)
magnitude = np.linalg.norm(v1) #  $\sqrt{1^2 + 2^2 + 3^2} = \sqrt{14}$ 

# Normalize
normalized = v1 / np.linalg.norm(v1)

# Element-wise operations
v3 = v1 * v2 # [4, 10, 18] (not dot product)

# Distance
distance = np.linalg.norm(v1 - v2)
```

Practical ML Example: k-NN

```
python
```

```
import numpy as np

# Training data (each row is a sample vector)
X_train = np.array([
    [1, 2],
    [2, 3],
    [10, 11],
    [11, 12]
])

# New point to classify
x_new = np.array([2, 1])

# Compute distances to all training vectors
distances = np.linalg.norm(X_train - x_new, axis=1)
# axis=1 means compute norm for each row

# k=2 nearest neighbors
k = 2
nearest_indices = np.argsort(distances)[:k]

print(f'Nearest neighbors: {X_train[nearest_indices]}')
```

Text Similarity Example: Cosine Distance

```
python
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.spatial.distance import cosine

documents = [
    "machine learning is great",
    "machine learning is awesome",
    "cooking recipes are nice"
]

# Convert to vectors
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(documents)

# Convert to dense arrays
v1 = X[0].toarray()[0]
v2 = X[1].toarray()[0]
v3 = X[2].toarray()[0]

# Cosine similarity
sim_1_2 = 1 - cosine(v1, v2) # ≈ 0.9 (similar)
sim_1_3 = 1 - cosine(v1, v3) # ≈ 0.1 (different)

print(f"Document 1 and 2 similarity: {sim_1_2}")
print(f"Document 1 and 3 similarity: {sim_1_3}")
```

Gradient Descent with Vectors

```
python
```

```

import numpy as np

# Synthetic data
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5]])
y = np.array([5, 7, 9, 11])

# Initialize weight vector
w = np.array([0.0, 0.0])
b = 0.0
learning_rate = 0.01
epochs = 100

# Gradient descent
for epoch in range(epochs):
    # Predictions:  $y_{pred} = X \cdot w + b$ 
    y_pred = np.dot(X, w) + b

    # Error
    error = y_pred - y

    # Gradient vectors
    dw = (2/len(X)) * np.dot(X.T, error) # Vector of gradients
    db = (2/len(X)) * np.sum(error)

    # Update weights (vector operation)
    w = w - learning_rate * dw
    b = b - learning_rate * db

    if epoch % 20 == 0:
        loss = np.mean(error ** 2)
        print(f"Epoch {epoch}, Loss: {loss:.4f}, w: {w}")

print(f"Final weights: {w}")

```

Embedding Vectors in NLP

python

```

from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np

texts = [
    "deep learning with neural networks",
    "machine learning algorithms",
    "neural networks for classification"
]

# TF-IDF Vectorization (converts text to vectors)
vectorizer = TfidfVectorizer(max_features=10)
vectors = vectorizer.fit_transform(texts).toarray()

print("Feature names (vocabulary):")
print(vectorizer.get_feature_names_out())

print("\nVector shape: ", vectors.shape) # (3 texts, 10 features)
print("\nFirst document vector:")
print(vectors[0])

# Similarity between documents (dot product after normalization)
# Or use cosine similarity
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vectors)
print("\nSimilarity matrix:")
print(similarity)

```

PCA Using Vectors

```
python
```

```

from sklearn.decomposition import PCA
import numpy as np

# Data: 100 samples, 50 features (50D vectors)
X = np.random.randn(100, 50)

# PCA to reduce to 2D
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

print(f"Original shape: {X.shape}")
print(f"Reduced shape: {X_reduced.shape}")

# Principal components are vectors
print(f"\nFirst principal component (50D vector):")
print(pca.components_[0])

print(f"\nExplained variance ratio: {pca.explained_variance_ratio_}")

# Reconstruction (project back to original space)
X_reconstructed = pca.inverse_transform(X_reduced)
print(f"\nReconstructed shape: {X_reconstructed.shape}")

```

Part 7: Advanced Vector Concepts

Gram-Schmidt Orthogonalization

Convert linearly independent vectors into orthonormal vectors.

Process:

Given vectors: v_1, v_2, v_3

1. $u_1 = v_1 / \|v_1\|$
2. $u_2 = (v_2 - (v_2 \cdot u_1)u_1) / \|(v_2 - (v_2 \cdot u_1)u_1)\|$
3. $u_3 = (v_3 - (v_3 \cdot u_1)u_1 - (v_3 \cdot u_2)u_2) / \|(v_3 - (v_3 \cdot u_1)u_1 - (v_3 \cdot u_2)u_2)\|$

Result: orthonormal vectors u_1, u_2, u_3

Vector Spaces and Subspaces

Span: All linear combinations of a set of vectors

span({[1,0], [0,1]}) = \mathbb{R}^2 (all 2D vectors)
span({[1,0]}) = all vectors of form [x, 0] (x-axis)

Subspace: A subset of a vector space that forms a vector space itself

The plane $z=0$ in \mathbb{R}^3 is a subspace
All vectors of form [x, y, 0]

Gram Matrix

Matrix of inner products between vectors.

Definition:

$G_{ij} = v_i \cdot v_j$

Example:

$v1 = [1, 0]$
 $v2 = [1, 1]$

 $G = \begin{bmatrix} v1 \cdot v1 & v1 \cdot v2 \\ v2 \cdot v1 & v2 \cdot v2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$

Part 8: Vector Applications Summary Table

Application	Vector Use	Key Concept
Feature representation	Each data point is a vector	Dimension = number of features
Distance metrics	Compute	
k-NN clustering	Find closest vectors	Euclidean/cosine distance
Linear regression	$y = X \cdot w$ (X is matrix of vectors)	Dot product for prediction

Application	Vector Use	Key Concept
Gradient descent	$w = w - \alpha \nabla L$ (∇L is gradient vector)	Direction of steepest descent
Word embeddings	Words as high-D vectors	Semantic relationships
PCA	Project onto principal component vectors	Dimensionality reduction
Attention (Transformers)	$q \cdot k$ (dot product of vectors)	Computing relevance weights
Recommendation	User/item similarity vectors	Cosine similarity
Anomaly detection	Distance from normal vectors	Outlier identification
CNN features	Activation vectors at each layer	Learned representations

Part 9: Vector Algebra Properties Reference

Addition Properties

$u + v = v + u$	(commutative)
$(u + v) + w = u + (v + w)$	(associative)
$u + 0 = u$	(identity)
$u + (-u) = 0$	(inverse)

Scalar Multiplication Properties

$\alpha(u + v) = \alpha u + \alpha v$	(distributive over vector addition)
$(\alpha + \beta)u = \alpha u + \beta u$	(distributive over scalar addition)
$\alpha(\beta u) = (\alpha\beta)u$	(associative)
$1u = u$	(identity)

Dot Product Properties

$u \cdot v = v \cdot u$	(commutative)
$u \cdot (v + w) = u \cdot v + u \cdot w$	(distributive)
$(\alpha u) \cdot v = \alpha(u \cdot v)$	(scalar multiplication)
$v \cdot v = \ v\ ^2$	(relationship to norm)
$u \cdot v = \ u\ \ v\ \cos(\theta)$	(geometric interpretation)

Norm Properties

$\|\alpha v\| = |\alpha| \|v\|$ (scalar property)

$\|u + v\| \leq \|u\| + \|v\|$ (triangle inequality)

$\|u\| \geq 0$ and $\|u\| = 0$ iff $u=0$ (positivity)

$\|u - v\|$ = distance between u and v

Part 10: Complete Examples

Example 1: Movie Recommendation Using Vectors

```
python
```

```

import numpy as np
from scipy.spatial.distance import cosine

# User vectors (5 movies: Action, Comedy, Drama, Thriller, SciFi)
user_alice = np.array([5, 2, 4, 3, 2]) # Likes action and drama
user_bob = np.array([5, 2, 4, 4, 1]) # Similar to Alice
user_charlie = np.array([1, 5, 2, 1, 5]) # Likes comedy and scifi

# Movie vectors (user ratings)
movie_1 = np.array([5, 1, 1, 1, 1]) # Action movie
movie_2 = np.array([1, 5, 1, 1, 1]) # Comedy movie
movie_3 = np.array([1, 1, 5, 1, 1]) # Drama movie

# Cosine similarity between users
def cosine_sim(u, v):
    return 1 - cosine(u, v)

sim_alice_bob = cosine_sim(user_alice, user_bob)
sim_alice_charlie = cosine_sim(user_alice, user_charlie)

print(f"Similarity Alice-Bob: {sim_alice_bob:.3f}")
print(f"Similarity Alice-Charlie: {sim_alice_charlie:.3f}")

# Alice should get recommendations from Bob (more similar)

# Recommendation scores (dot product with user vector)
alice_score_movie1 = np.dot(user_alice, movie_1)
alice_score_movie2 = np.dot(user_alice, movie_2)
alice_score_movie3 = np.dot(user_alice, movie_3)

print(f"\nAlice's predicted scores:")
print(f"Movie 1 (Action): {alice_score_movie1}")
print(f"Movie 2 (Comedy): {alice_score_movie2}")
print(f"Movie 3 (Drama): {alice_score_movie3}")

# Recommend highest-scoring unwatched movies

```

Example 2: Semantic Search Using Vector Similarity

```
python
```

```

import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.spatial.distance import cosine

# Documents
documents = [
    "machine learning algorithms process data",
    "deep neural networks learn representations",
    "cooking recipes require ingredients",
    "vegetables are healthy food",
    "artificial intelligence and machine learning"
]

# Convert to vectors
vectorizer = TfidfVectorizer(max_features=20)
doc_vectors = vectorizer.fit_transform(documents).toarray()

# Query
query = "machine learning"
query_vector = vectorizer.transform([query]).toarray()[0]

# Find most similar documents
similarities = [1 - cosine(query_vector, doc) for doc in doc_vectors]
top_indices = np.argsort(similarities)[::-1][:3]

print("Query: 'machine learning'")
print("\nTop 3 similar documents:")
for idx in top_indices:
    print(f"{idx+1}. {documents[idx]} (similarity: {similarities[idx]:.3f})")

```

Example 3: Clustering with k-Means (Vector Perspective)

```
python
```

```

import numpy as np
from sklearn.cluster import KMeans

# Data points (vectors)
X = np.array([
    [1, 1],
    [1, 2],
    [2, 1],
    [10, 10],
    [10, 11],
    [11, 10]
])

# k-means clustering
kmeans = KMeans(n_clusters=2, random_state=42)
labels = kmeans.fit_predict(X)

print("Data points (vectors):")
for i, point in enumerate(X):
    print(f"Point {i}: {point} → Cluster {labels[i]}")

print("\nCluster centers (vectors):")
for i, center in enumerate(kmeans.cluster_centers_):
    print(f"Cluster {i} center: {center}")

# Each point is a 2D vector
# Distance to cluster centers determines assignment
# Centers are also vectors

```

Example 4: Neural Network Forward Pass (Vectors)

```
python
```

```

import numpy as np

# Simplified neural network
def relu(x):
    return np.maximum(0, x)

def softmax(x):
    exp_x = np.exp(x - np.max(x)) # Numerical stability
    return exp_x / np.sum(exp_x)

# Network parameters (weight vectors/matrices)
W1 = np.random.randn(784, 128) # 784 input features → 128 hidden
b1 = np.zeros(128)
W2 = np.random.randn(128, 10) # 128 hidden → 10 output classes
b2 = np.zeros(10)

# Input image vector (flattened 28×28)
x = np.random.randn(784)

print(f"Input vector shape: {x.shape}")

# Forward pass (all vector operations)
# Hidden layer:  $y1 = \text{relu}(x \cdot W1 + b1)$ 
z1 = np.dot(x, W1) + b1 # Dot product:  $784D \cdot 784 \times 128 = 128D$ 
a1 = relu(z1)

print(f"Hidden layer vector shape: {a1.shape}")

# Output layer:  $y2 = \text{softmax}(a1 \cdot W2 + b2)$ 
z2 = np.dot(a1, W2) + b2 # Dot product:  $128D \cdot 128 \times 10 = 10D$ 
output = softmax(z2)

print(f"Output vector shape: {output.shape}")
print(f"Output probabilities: {output}")
print(f"Predicted class: {np.argmax(output)}")

# Every computation involves vectors!

```

Example 5: Vector Projection in Feature Selection

```
python
```

```

import numpy as np
from sklearn.preprocessing import StandardScaler

# Feature vectors (each column is a feature vector)
X = np.array([
    [1, 2, 3],
    [2, 3, 4],
    [3, 4, 5],
    [4, 5, 6],
    [5, 6, 7]
])

# Target vector
y = np.array([2, 4, 6, 8, 10])

# Standardize
X_scaled = StandardScaler().fit_transform(X)

print("Feature vectors:")
for i in range(X.shape[1]):
    feature_vector = X_scaled[:, i]

    # Project target onto feature vector
    projection_length = np.dot(y, feature_vector) / np.linalg.norm(feature_vector)

    print(f"Feature {i}: projection onto target = {projection_length:.3f}")

# Larger projection → feature more aligned with target
# Use this for feature selection

```

Part 11: Key Takeaways

Why Vectors Matter

1. **Data representation:** Every ML problem starts with vectors
2. **Efficient computation:** Vector operations are highly optimized on modern hardware
3. **Geometric intuition:** Vectors provide visual understanding of high-dimensional data
4. **Mathematical foundation:** Linear algebra is fundamental to ML
5. **Scalability:** Vectorized code scales to millions of data points

Vector Concepts in ML

Concept	Purpose	Example
Dot product	Similarity/prediction	Neural network: $x \cdot w$
Magnitude	Size/distance	Regularization, normalization
Direction	Orientation/angle	Gradient descent direction
Projection	Component extraction	Feature importance
Normalization	Scale invariance	Cosine similarity
Orthogonality	Independence	Basis vectors in PCA

Best Practices

- Always check vector dimensions before operations
- Use vectorized operations (NumPy) instead of loops
- Normalize vectors when computing similarity (especially cosine)
- Think geometrically: distances, angles, projections
- Understand that each data point is a vector in feature space
- Use vectors to represent learned representations (embeddings)
- Remember: high-dimensional space intuition fails—use mathematics

From Vectors to Matrices to Tensors

Scalar: Single number (0D)
Vector: List of numbers (1D) — [1, 2, 3]
Matrix: 2D array (2D) — [[1,2],[3,4]]
Tensor: nD array (3D+) — [[[1,2],[3,4]],[[5,6],[7,8]]]

In ML:

- Vectors: Individual samples, features, parameters
- Matrices: Datasets, weights, covariance
- Tensors: Images (3D), videos (4D), batches of data

Part 12: Practice Problems

- Given vectors $u = [3, 4]$ and $v = [1, 2]$, compute:
 - $u + v$
 - $u \cdot v$
 - $\|u\|$ and $\|v\|$
 - Angle between u and v
 - Cosine similarity
 - A dataset has 100 images, each 28×28 pixels. How many dimensions when represented as vectors?
 - In k-NN with $k=5$, how many distance computations are needed for 1000 training vectors and 100 test vectors?
 - Why is cosine similarity better than Euclidean distance for text data?
 - In gradient descent, if gradient vector $\nabla L = [2, -3, 1]$, which direction should weights move?
 - Two embedding vectors have dot product 0. What does this tell you?
 - How does vector normalization affect cosine similarity computation?
 - In collaborative filtering, what does the dot product of user and item vectors represent?
-

Quick Reference

```
python

# Essential vector operations (NumPy)
import numpy as np

u, v = np.array([1, 2, 3]), np.array([4, 5, 6])

u + v      # Addition
u - v      # Subtraction
2 * u      # Scalar multiplication
np.dot(u, v) # Dot product
np.linalg.norm(u) # Magnitude
u / np.linalg.norm(u) # Normalize
np.linalg.norm(u - v) # Euclidean distance
np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v)) # Cosine
np.cross(u, v) # Cross product (3D only)
np.arccos(np.dot(u, v) / (np.linalg.norm(u) * np.linalg.norm(v))) # Angle
```

This comprehensive vector tutorial covers fundamentals through advanced ML applications with code examples and practical insights!