Linear Algebra for Machine Learning Beginners:
A Super Simple Guide to Essential Ideas

Saurabh

Abstract

Welcome! This guide is designed to make some important ideas from linear algebra feel easy and natural, especially if you're new to Machine Learning (ML) or if it's been a while since you've looked at these math concepts. We'll use lots of simple examples and real-world analogies to explain things like vector spaces, and why knowing if vectors are "independent" is a big deal in ML. Think of this as your friendly starting point to build confidence and understanding. (This version is more concise and focuses on core ideas for beginners).

Contents

A Quick Note on Mathematical "Rules of the Game"

The Big Picture Idea 0.1: Why Formal Rules?

You might wonder why mathematicians create strict rules for concepts like "groups" or "vector spaces." Think of it like grammar for a language or the rules for a board game. These rules ensure that everyone understands things the same way and that operations (like adding numbers or combining ideas) work predictably and consistently. In computer science, and especially in Machine Learning, this predictability is crucial. Computers need exact instructions. When we build ML models, the underlying math needs to be solid and follow consistent rules so the models can learn reliably and give dependable results. We won't dive deep into all abstract structures here, but knowing that these foundations exist helps appreciate why ML isn't just magic—it's built on logical principles!

The Key Rules (Formal Definition) 0.2: Basic Idea of a "Group" (Highly Simplified)

Very briefly, a mathematical **group** involves:

1. A collection of items.
2. One way to combine any two items.

This setup must obey a few key properties:

- **Closure:** Combining items keeps you within the collection.
- **Associativity:** For three or more items, the grouping of combinations doesn't change the final result (e.g., $(a \otimes b) \otimes c = a \otimes (b \otimes c)$).
- **Identity:** There's a special item that does nothing when combined (like 0 in addition, or 1 in multiplication).
- **Inverse:** Every item has a partner that "undoes" it, bringing you back to the identity item.

If the order of combination doesn't matter (e.g., $a \otimes b = b \otimes a$), the group is called **Abelian**. *Main takeaway: These rules create a well-behaved system. Now, let's move to something more directly used in ML: Vector Spaces!*

1 Vector Spaces: The Playground for Our Data

The Big Picture Idea 1.1: What's a Vector Space?

Imagine a special 'playground' for 'vectors.' Vectors are just lists of numbers that describe something. For example, '[height, weight]' or '[pixel1_brightness, pixel2_brightness, ...]'. **In Machine Learning, data is almost always turned into these vector lists for computers to process.**

In a Vector Space playground, you can do two main things:

1. **Add vectors**: Add two lists of numbers to get a new list.
2. **Scale vectors**: Multiply a list by a single number (a 'scalar') to stretch, shrink, or flip it.

For this playground to be official, these operations must follow 10 sensible rules (axioms) that ensure everything is consistent and predictable—vital for reliable ML algorithms.

Analogy Corner: What's this like in ML? 1.2: Fruit Feature Space

Let's say we describe fruits with vectors: an apple's vector $\mathbf{v}_{\text{apple}} = [\text{redness}, \text{roundness}, \text{sweetness}]$. This is a vector in a 3D 'fruit space.'

- **Adding**: Averaging apple and pear vectors might give a 'generic pome fruit' vector, still in this fruit space.
- **Scaling**: If an ML model finds 'sweetness' very important, it might internally multiply that feature, still resulting in a valid vector in the fruit space.

This 'fruit space,' with consistent operations, is a vector space. ML models find patterns here.

The Key Rules (Formal Definition) 1.3: The 10 Vector Space Rules (The Gist)

A set of vectors V, with vector addition $(+)$ and scalar multiplication $(\cdot)$, is a **vector space** if:

Vector Addition is well-behaved (forms an Abelian Group):

- $\mathbf{u} + \mathbf{v}$ is in V (Closure).
- $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$ (Commutativity).
- $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$ (Associativity).
- A **zero vector** $\mathbf{0}$ exists: $\mathbf{v} + \mathbf{0} = \mathbf{v}$.
- Every $\mathbf{v}$ has an inverse $-\mathbf{v}$: $\mathbf{v} + (-\mathbf{v}) = \mathbf{0}$.

Scalar Multiplication works well with addition:

- $c \cdot \mathbf{v}$ is in V (Closure).
- $c \cdot (\mathbf{u} + \mathbf{v}) = c \cdot \mathbf{u} + c \cdot \mathbf{v}$ (Distributive 1).
- $(c + d) \cdot \mathbf{v} = c \cdot \mathbf{v} + d \cdot \mathbf{v}$ (Distributive 2).
- $c \cdot (d \cdot \mathbf{v}) = (cd) \cdot \mathbf{v}$ (Associativity of scalars).
- $1 \cdot \mathbf{v} = \mathbf{v}$ (Scalar identity).

Key idea: These rules ensure vectors add and scale consistently, which is critical for ML.

Let's See an Example 1.4: Common Vector Spaces

$\mathbb{R}^2$ (pairs of numbers (x, y), like 2 features) with standard addition and scalar multiplication is a vector space.

The set of 2×2 matrices also forms a vector space, where each matrix $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ is a "vector." Addition is entry-wise, and scalar multiplication multiplies all entries. The zero matrix is the "zero vector."

Zoom in on Machine Learning 1.5: Matrices in ML

Datasets are often matrices (samples $\times$ features). Neural network layers involve matrix math. The fact these live in vector spaces means operations like combining data batches or adjusting model weights (often matrices) are mathematically sound.

2 Vector Subspaces: Simpler Views of Data

The Big Picture Idea 2.1: Smaller Playgrounds

Within a large vector space (e.g., data with 100s of features), you can often find a smaller part that's still a complete vector space itself. This is a "subspace."

Analogy Corner: What's this like in ML? 2.2: PCA and Subspaces

If your customer data is in $\mathbb{R}^{100}$ (100 features), PCA might find that most important variations lie on a 2D plane (a subspace). Working in this simpler 2D subspace can make ML models faster and less prone to noise, because it captures the essence. This 2D plane is a subspace: adding or scaling vectors on this plane keeps them on the plane, and the origin is included.

The Key Rules (Formal Definition) 2.3: Quick Subspace Check

A part U of a vector space V is a **subspace** if:

1. $\mathbf{0}$ (of V) is in U. (Origin must be there.)
2. If $\mathbf{u}, \mathbf{w}$ are in U, then $\mathbf{u} + \mathbf{w}$ is in U. (Closed under addition.)
3. If $\mathbf{u}$ is in U and c is a scalar, then $c \cdot \mathbf{u}$ is in U. (Closed under scalar multiplication.)

Let's See an Example 2.4: Is the x-axis a Subspace of $\mathbb{R}^2$?

Yes. The x-axis is vectors $(x, 0)$.

1. $(0, 0)$ is on the x-axis. (Contains zero)
2. $(x_1, 0) + (x_2, 0) = (x_1 + x_2, 0)$, which is on the x-axis. (Closed under addition)
3. $c \cdot (x_1, 0) = (cx_1, 0)$, which is on the x-axis. (Closed under scaling)

A line $y = x + 1$ is NOT a subspace because it doesn't contain $(0, 0)$.

Zoom in on Machine Learning 2.5: Why Subspaces Matter for Data Simplification

When ML techniques like PCA simplify data, they often project it onto a subspace. This subspace must include the origin (if data is centered) to avoid weird shifts and maintain a meaningful representation of the 'average' or 'baseline' data point.

3 Linear Combinations: Mixing Your Feature Ingredients

The Big Picture Idea 3.1: The Basic Recipe

A "linear combination" is making a new vector by taking some starting vectors, scaling each one (multiplying by a number), and then adding the scaled versions together. **This is how many ML models make predictions and how new data features can be created.**

Analogy Corner: What's this like in ML? 3.2: Predicting with Weighted Features

An ML model predicting exam pass likelihood might use features: $\mathbf{v}_{\text{study}}$ (hours studied) and $\mathbf{v}_{\text{grades}}$ (past grades). It learns weights (scalars) w_1, w_2. The prediction score is a linear combination: Score $= w_1\mathbf{v}_{\text{study}} + w_2\mathbf{v}_{\text{grades}}$. The model 'mixes' features, adjusted by their learned importance.

The Key Rules (Formal Definition) 3.3: The Formal Recipe

Given vectors $\mathbf{v}_1, ..., \mathbf{v}_k$, a **linear combination** is any vector $\mathbf{w}$ of the form:

$$\mathbf{w} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + ... + c_k\mathbf{v}_k$$

where $c_1, ..., c_k$ are scalars (the 'weights' or 'amounts'). In ML, these scalars are often what the model learns.

Let's See an Example 3.4: Making a 2D Vector

Let $\mathbf{v}_1 = (1, 0)$ and $\mathbf{v}_2 = (0, 1)$. Can we get $\mathbf{w} = (3, 4)$? Yes: $3\mathbf{v}_1 + 4\mathbf{v}_2 = 3(1, 0) + 4(0, 1) = (3, 0) + (0, 4) = (3, 4)$. So $(3, 4)$ is a linear combination of $(1, 0)$ and $(0, 1)$.

Let's See an Example 3.5: When It's Impossible

Let $\mathbf{u} = (1, 1)$ and $\mathbf{v} = (2, 2)$. Can we get $\mathbf{z} = (3, 5)$? We need $c_1(1, 1) + c_2(2, 2) = (3, 5)$, so $(c_1 + 2c_2, c_1 + 2c_2) = (3, 5)$. This requires $c_1 + 2c_2 = 3$ AND $c_1 + 2c_2 = 5$, which is impossible. So, $(3, 5)$ is NOT a linear combination of $(1, 1)$ and $(2, 2)$ (because $\mathbf{u}$ and $\mathbf{v}$ point in the same direction).

Quick Check: Why it Matters for ML 3.6: Span: What Your Features Can Describe

The "span" of a set of vectors is **all possible vectors** you can create using all their linear combinations. It's the entire 'reach' of your features.

- Span of just $(1, 0)$ in $\mathbb{R}^2$ is the x-axis.
- Span of $(1, 0)$ and $(0, 1)$ in $\mathbb{R}^2$ is the whole $\mathbb{R}^2$ plane.
- Span of $(1, 1)$ and $(2, 2)$ in $\mathbb{R}^2$ is just the line $y = x$.

The span of vectors always forms a subspace.

Zoom in on Machine Learning 3.7: The Power of Your Features (Span)

The span of your ML model's input features defines what it can possibly learn or represent. If a real-world pattern is outside this span, your model can't capture it. Good feature engineering aims to ensure your features span the important aspects of the problem.

4 Linear Independence: Are Features Redundant?

The Big Picture Idea 4.1: Unique vs. Echoes

With a set of vectors (features), are they all contributing unique info, or are some just 'echoes' (linear combinations) of others?

- **Linearly Independent**: All features are originals! None can be made by combining others. Each adds a new 'direction' of information. (Desirable for efficient ML models).
- **Linearly Dependent**: At least one feature is a 'copycat,' redundant. (Can confuse ML models, e.g., multicollinearity).

Analogy Corner: What's this like in ML? 4.2: Height in CM vs. Inches

Features: $\mathbf{f}_1$ (height in cm), $\mathbf{f}_2$ (weight in kg), $\mathbf{f}_3$ (height in inches).

- $\mathbf{f}_1, \mathbf{f}_2$ are likely linearly independent (different info).
- $\mathbf{f}_3$ is linearly dependent on $\mathbf{f}_1$ (since height in inches = height in cm / 2.54). $\mathbf{f}_3$ adds no new info if $\mathbf{f}_1$ is present.

ML prefers independent features for robustness and interpretability.

The Key Rules (Formal Definition) 4.3: The Zero Vector Test for Redundancy

To test if vectors $\{\mathbf{v}_1, ..., \mathbf{v}_k\}$ are linearly (in)dependent, check if their linear combination can be the **zero vector 0**:

$$c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + ... + c_k\mathbf{v}_k = \mathbf{0}$$

- **INDEPENDENT** if the ONLY solution is ALL $c_i = 0$ (trivial solution).
- **DEPENDENT** if there's a solution where AT LEAST ONE $c_i \neq 0$ (non-trivial solution). This means one vector can be written using others, hence redundant.

Let's See an Example 4.4: Basic 2D Vectors

Are $\mathbf{v}_1 = (1,0), \mathbf{v}_2 = (0,1)$ linearly independent? $c_1(1,0) + c_2(0,1) = (0,0) \implies (c_1, c_2) = (0,0)$. Only solution is $c_1 = 0, c_2 = 0$. Yes, they are **linearly independent**.

Let's See an Example 4.5: Three 2D Vectors

Are $\mathbf{u} = (1,1), \mathbf{v} = (2,2), \mathbf{w} = (3,0)$ linearly independent? (Intuition: 3 vectors in 2D space are likely dependent). Solve $c_1(1,1) + c_2(2,2) + c_3(3,0) = (0,0)$. This gives: $c_1 + 2c_2 + 3c_3 = 0$ and $c_1 + 2c_2 = 0$. From 2nd eqn: $c_1 = -2c_2$. Sub into 1st: $(-2c_2) + 2c_2 + 3c_3 = 0 \implies 3c_3 = 0 \implies c_3 = 0$. If $c_3 = 0$ and $c_1 = -2c_2$, pick $c_2 = 1 \implies c_1 = -2$. A non-trivial solution is $c_1 = -2, c_2 = 1, c_3 = 0$. So, they are **linearly dependent** (since $\mathbf{v} = 2\mathbf{u}$).

Quick Check: Why it Matters for ML 4.6: Key Points on Independence

- A set including the zero vector is always linearly **dependent**. (A zero feature is useless).
- Two vectors are linearly dependent iff one is a scalar multiple of the other (collinear).

5 Basis and Dimension: Your Data's Essential Toolkit

The Big Picture Idea 5.1: Core Building Blocks

For any vector space (like your data's feature space), we want the smallest set of "fundamental" vectors that can build *every other vector* in that space through linear combinations.

- This smallest fundamental set is a **basis**. (The most compact, non-redundant feature set capturing all essential info).
- The **number of vectors** in a basis is the **dimension** of the space. (How many independent 'directions' or pieces of information your data truly has).

Analogy Corner: What's this like in ML? 5.2: Describing Locations and Data Simply

- **Basis for a 2D map ($\mathbb{R}^2$)**: Needs two different direction vectors, like $\mathbf{i} = (1, 0)$ (East-West) and $\mathbf{j} = (0, 1)$ (North-South). This set $\{\mathbf{i}, \mathbf{j}\}$ is a basis.
- **Dimension of $\mathbb{R}^2$**: is 2 (from the 2 basis vectors).
- **PCA's Basis**: If PCA finds 5 principal components capture most variance in 100-feature data, these 5 components form a basis for an effective 5-dimensional subspace. The 'intrinsic dimension' might be 5.

The Key Rules (Formal Definition) 5.3: Two Must-Haves for a Basis

A set of vectors $B = \{\mathbf{v}_1, ..., \mathbf{v}_n\}$ is a **basis** for vector space V if:

1. **It Spans V**: Every vector in V can be written as a linear combination of vectors in B. (Complete coverage).

2. **It's Linearly Independent**: No vector in B is a linear combination of others in B. (No redundancy; most efficient).

Let's See an Example 5.4: Standard Basis for $\mathbb{R}^2$

Is $B = \{(1,0),(0,1)\}$ a basis for $\mathbb{R}^2$?

1. Spans $\mathbb{R}^2$? Yes, any $(x,y) = x(1,0) + y(0,1)$.
2. Linearly independent? Yes, $c_1(1,0)+c_2(0,1) = (0,0)$ only if $c_1 = c_2 = 0$.

So, B is a basis. Dimension of $\mathbb{R}^2$ is 2.

Let's See an Example 5.5: Not a Basis

Is $S = \{(1,1),(2,0),(0,2)\}$ a basis for $\mathbb{R}^2$?

1. Spans $\mathbb{R}^2$? Yes (e.g., $(2,0)$ and $(0,2)$ can already span it).
2. Linearly independent? No. We have 3 vectors in a 2D space. They must be dependent. (E.g., $2(1,1) - 1(2,0) - 1(0,2) = \mathbf{0}$).

Since S is linearly dependent, it's not a basis. It's not efficient. Removing $(0,2)$ would leave $\{(1,1),(2,0)\}$, which *is* a basis.

Quick Check: Why it Matters for ML 5.6: Dimension Facts

- The number of vectors in any basis for a space is always the same (its dimension).
- If dimension is n, any n linearly independent vectors form a basis.
- If dimension is n, any n vectors that span the space form a basis.

Zoom in on Machine Learning 5.7: Basis in ML Applications

Finding a good basis is key:

- **Dimensionality Reduction (PCA, Autoencoders)**: Aim to find a lower-dimensional basis (e.g., principal components, latent features) that captures the data's essence.
- **Feature Engineering/Selection**: Goal is to find features that are like a basis for the relevant 'information space' – powerful enough (span) and

efficient (independent).