Vector Spaces and Their Implementation with NumPy

Introduction

Vector spaces form the backbone of modern machine learning (ML), powering algorithms and allowing intuitive manipulation of data. This handbook gives a concise introduction to the theory of vector spaces, explains why they're crucial in ML, and demonstrates their practical use with NumPy for computational tasks.

1. Theoretical Foundation

1.1 What is Vector Space?

A **vector space** (or linear space) is a mathematical structure consisting of a set of elements called **vectors**, alongside two operations:

- **Vector addition:** The sum of two vectors yields another vector.
- **Scalar multiplication:** Multiplying a vector by a scalar (a number) results in another vector in the space.

A vector space over the real numbers $\mathbb{R}$ must satisfy certain axioms, such as associativity, commutativity, distributivity, and existence of additive identity and inverses.

1. Commutativity:

$$\mathbf{X} + \mathbf{Y} = \mathbf{Y} + \mathbf{X}.$$

2. Associativity of vector addition:

$$(\mathbf{X} + \mathbf{Y}) + \mathbf{Z} = \mathbf{X} + (\mathbf{Y} + \mathbf{Z}).$$

3. Additive identity: For all $\mathbf{X}$,

$$\mathbf{0} + \mathbf{X} = \mathbf{X} + \mathbf{0} = \mathbf{X}.$$

4. Existence of additive inverse: For any $\mathbf{X}$, there exists a $-\mathbf{X}$ such that

$$\mathbf{X} + (-\mathbf{X}) = \mathbf{0}.$$

5. Associativity of scalar multiplication:

$$r(s\,\mathbf{X}) = (r\,s)\,\mathbf{X}.$$

6. Distributivity of scalar sums:

$$(r + s)\,\mathbf{X} = r\,\mathbf{X} + s\,\mathbf{X}.$$

7. Distributivity of vector sums:

$$r(\mathbf{X} + \mathbf{Y}) = r\,\mathbf{X} + r\,\mathbf{Y}.$$

8. Scalar multiplication identity:

$$1\,\mathbf{X} = \mathbf{X}.$$

1.2 Key Concepts

- **Basis:** A set of linearly independent vectors that spans the entire vector space.
- **Dimension:** The number of vectors in a basis, representing the space's degrees of freedom.
- **Subspace:** A subset of a vector space that is also a vector space.
- **Linear Independence:** Vectors that cannot be written as a linear combination of others.
- **Span:** The set of all possible linear combinations of a set of vectors.

1.3 Relevance in Machine Learning

- **Data Representation:** Features, images, and signals are often high-dimensional vectors.
- **Transformations:** Operations like PCA, embeddings, and neural network layers are all linear (or non-linear) transformations in vector spaces.
- **Optimization:** Many ML algorithms (e.g., gradient descent) operate in vector spaces to minimize loss.

2. Vector Spaces in Machine Learning

2.1 Typical Uses

- **Feature Vectors:** Each data point's features comprise a vector in a high-dimensional space.
- **Embeddings:** NLP and vision models map words or images to numeric vector spaces.
- **Principal Component Analysis (PCA):** Projects data vectors onto lower-dimensional subspaces while preserving variance.

2.2 Computational Operations

- **Distance calculation** (e.g., Euclidean, cosine)
- **Projections** and **orthogonality** checks
- **Dot products** and **matrix multiplications**

3. Implementing Vector Space Operations with NumPy

3.1 Creating and Manipulating Vectors

```
import numpy as np

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

# Vector addition
v_sum = v1 + v2  # array([3, 7])




# Scalar multiplication
v_scaled = 2 * v1  # array([4, 6])
```

3.2 Dot Product and Norm

```
# Dot product
dot = np.dot(v1, v2)  # 2*1 + 3*4 = 14

# Norm (magnitude)
norm = np.linalg.norm(v1)  # sqrt(2^2 + 3^2) ≈ 3.6056
```

3.3 Orthogonality and Projections

```
# Orthogonality (dot product == 0)
is_orthogonal = np.dot(v1, v2) == 0

# Projection of v1 onto v2
proj = (np.dot(v1, v2) / np.dot(v2, v2)) * v2
```

3.4 Working with Matrices (Higher-Dimensional Spaces)

Matrices can represent sets of vectors (as rows or columns):

```
# Matrix of vectors (each row is a vector)
A = np.array([[1, 2], [3, 4], [5, 6]])

# Linear combination: c1*A[0] + c2*A[1]
c1, c2 = 1.5, -0.5
```

```
linear_combo = c1*A[0] + c2*A[1]
```

3.5 Dimensionality Reduction: Example with PCA

```
from sklearn.decomposition import PCA

# Sample data (m samples x n features)
X = np.random.rand(5, 3)

# Project to 2D vector space
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
```

4. Theoretical and Computational Considerations

4.1 Linear Independence and Basis with NumPy

Check if vectors are linearly independent:

```
# Stack vectors as columns
M = np.stack([v1, v2], axis=1)
rank = np.linalg.matrix_rank(M)
is_independent = rank == M.shape[1]
```

4.2 Computational Efficiency

- **Vectorized operations (NumPy):** Fast, memory-efficient for large ML datasets.
- **Matrix Multiplications:** Central to training neural networks and transforming data.

5. Further Reading and Practice

- Linear Algebra resources for ML
- NumPy tutorials and documentation
- Applied linear algebra tasks: PCA, SVD, embeddings

References

"Deep Learning Book" by Ian Goodfellow, et al.

NumPy official documentation.