Mathematical Foundation for ML

Q. Explain the concept of inner product and its significance in ML

The **inner product**, also called the **dot product**, is a **mathematical operation** that takes two vectors and returns a **single scalar value**.

For two vectors $\vec{a} = [a_1, a_2, ..., a_n]$ and $\vec{b} = [b_1, b_2, ..., b_n]$:

$$\vec{a} \cdot \vec{b} = a_1b_1 + a_2b_2 + \cdots + a_nb_n$$

Geometric Interpretation:

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

Where:

- ||a⃗|| = magnitude (length) of vector a⃗\vec{a}a
- θ = angle between vectors

If vectors point in the same direction, inner product is **maximum**
If vectors are orthogonal (90°), inner product is **zero**

Example Calculation:

Let $\vec{a} = [2, 3]$ and $\vec{b} = [4, 1]$

$$\vec{a} \cdot \vec{b} = (2)(4) + (3)(1) = 8 + 3 = 11$$

Significance of Inner Product in Machine Learning:

---

1. Similarity Measurement

- Measures **how similar two vectors are** in direction.
- Commonly used in **text classification** (e.g., cosine similarity for documents).

If inner product is large → vectors are similar (e.g., similar documents)

---

2. In Linear Classifiers (e.g., SVM, Perceptron)

- Decision function:

$$f(x) = \vec{w} \cdot \vec{x} + b$$

- Here, inner product between **weight vector $\vec{w}$** and **input vector $\vec{x}$** determines classification output.

Used to compute how far a point is from the decision boundary.

---

3. In Neural Networks

- Neuron output = Weighted sum of inputs = **dot product of weights and input vector**

$$z = \vec{w} \cdot \vec{x} + b$$

- Followed by activation function: y=f(z)

---

4. In Kernels and SVM

- **Kernel functions** compute inner products in high-dimensional space without explicitly transforming the data.
- Example:

$$K(x, y) = \phi(x) \cdot \phi(y)$$

---

5. Projection and Orthogonality

- Used to **project one vector onto another**.
- If $\vec{a} \cdot \vec{b} = 0$, then vectors are **orthogonal** (perpendicular), meaning they are **uncorrelated** in ML.

Q. Describe the types of norms and their significance in Machine learning

A **norm** is a function that measures the **size or length of a vector** in a vector space.

Mathematically, a norm maps a vector $\vec{x}$ to a **non-negative scalar**:

$$||\vec{x}|| \geq 0$$

In ML, norms help measure **distances, similarities, and complexity** of models and data points.

Types of Norms:

There are several norms used in ML, the most important being:

---

1. L1 Norm (Manhattan Norm / Taxicab Norm)

$$||\vec{x}||_1 = \sum_{i=1}^{n} |x_i|$$

- **Sum of absolute values** of vector components
- Used in **Lasso Regression** (adds sparsity — reduces features)

---

2. L2 Norm (Euclidean Norm)

$$||\vec{x}||_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$$

- **Straight-line distance** from origin to point
- Used in **Ridge Regression** (smooth shrinkage of weights)
- Also used to **normalize vectors** in distance-based algorithms

---

3. Lp Norm (General Form)

$$||\vec{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}$$

- A **generalized norm** — L1 and L2 are just special cases where p=1p=1 or p=2p=2

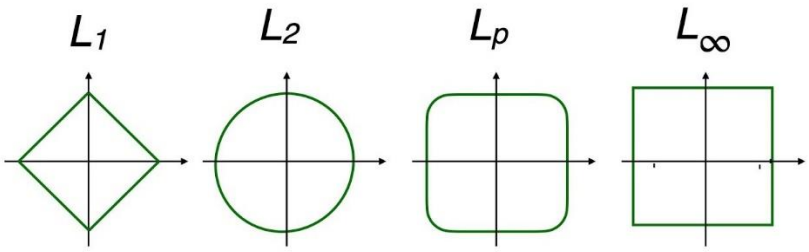
---

4. Max Norm (L∞ Norm)

$$||\vec{x}||_\infty = \max(|x_1|, |x_2|, ..., |x_n|)$$

- Takes the **maximum absolute value** among all vector elements
- Useful in **adversarial settings** (worst-case scenario)

Also called **Chebyshev norm**



Significance of Norms in Machine Learning:

---

1. Regularization (Controlling Overfitting)

- **L1 Norm (Lasso):** Encourages sparsity in models by shrinking irrelevant weights to **zero**
- **L2 Norm (Ridge):** Penalizes large weights smoothly, improving generalization

---

2. Distance Measures

- Norms are used to compute distances in:
  - **k-NN algorithms**
  - **Clustering (e.g., K-Means)**
  - **Similarity matching**

---

3. Vector Normalization

- Normalize feature vectors using L2 norm to convert them into **unit vectors**, which helps:
  - Improve gradient descent
  - Avoid scale issues

---

4. Optimization

- Norms appear in **loss functions** (like MSE = L2 norm squared)
- They're used in **gradient calculations** and convergence checks

Q. What are orthogonal vectors? Give an example

Two vectors are said to be **orthogonal** if the **angle between them is 90 degrees (π/2 radians)**.

In mathematical terms, two vectors $\vec{a}$ and $\vec{b}$ are **orthogonal** if:

$$\vec{a} \cdot \vec{b} = 0$$

That is, their **dot product is zero**.

Geometric Meaning: Orthogonal vectors are **perpendicular** to each other in space.

Example:

Let:

$$\vec{a} = [1, 2], \vec{b} = [2, -1]$$

Now compute their dot product:

$$\vec{a} \cdot \vec{b} = (1)(2) + (2)(-1) = 2 - 2 = 0$$

Since the dot product is 0, **these vectors are orthogonal**.

Properties of Orthogonal Vectors:

- If $\vec{a} \cdot \vec{b} = 0$, the vectors are **linearly independent**.
- Orthogonal vectors form the basis for **orthonormal systems** (used in PCA, SVD).
- Important in **dimensionality reduction**, **signal processing**, and **projection** tasks in ML.

Q. What are orthogonal projections and their use in ML

An **orthogonal projection** is a way to **project a vector onto another vector or subspace**, such that the resulting projection is **perpendicular (orthogonal)** to the surface it's being projected onto.

In simple terms: **Drop a shadow of a vector onto a surface** such that the shadow lies flat and is perpendicular to the drop line.

🔧 **Mathematical Formula (1D Projection):**

Let $\vec{a}$ be the vector to be projected onto $\vec{b}$. The **orthogonal projection** of $\vec{a}$ onto $\vec{b}$ is:

$$\text{proj}_{\vec{b}}(\vec{a}) = \frac{\vec{a} \cdot \vec{b}}{||\vec{b}||^2} \cdot \vec{b}$$

Where:

- $\vec{a} \cdot \vec{b}$ = dot product of $\vec{a}$ and $\vec{b}$
- $||\vec{b}||$ = norm (length) of $\vec{b}$

Use of Orthogonal Projections in Machine Learning:

---

1. Dimensionality Reduction (e.g., PCA)

- In **Principal Component Analysis**, data is projected onto **principal components** (orthogonal axes).
- This reduces dimensions while preserving the most variance (information) in the data.

---

2. Feature Transformation

- Used to **project data into new feature spaces** (like in **LDA** or **SVM kernel methods**) for better separation of classes.

---

3. Least Squares Regression

- The predicted output in linear regression is an **orthogonal projection** of the target vector onto the subspace spanned by input vectors.

$$\hat{y} = Xw = \text{proj}_{\text{col}(X)}(y)$$

- You're projecting the target values onto the space that can be expressed by your input features.

---

4. Orthogonalization in Neural Nets / Optimization

- Helps in **decorrelating features** or gradients.
- Improves **training stability** and **model interpretability**.

Q. Define eigenvalues and eigenvectors. Explain their role in ML

For a **square matrix** A, if there exists a **non-zero vector** $\vec{v}$ such that:

$$A\vec{v} = \lambda\vec{v}$$

Then:

- $\vec{v}$ is called an **eigenvector** of matrix A
- λ is the corresponding **eigenvalue**

Meaning:

- When matrix A acts on vector $\vec{v}$, it **does not change the direction** of $\vec{v}$, only **scales it by factor** λ.
- So, eigenvectors show the **"natural directions"** along which transformation by A acts as **pure scaling**.

Mathematical Condition:

To find eigenvalues, solve the **characteristic equation**:

$$\det(A - \lambda I) = 0$$

Where:

- λ = eigenvalue
- I = identity matrix
- det = determinant

Once λ is known, solve:

$$(A - \lambda I)\vec{v} = 0$$

to find the corresponding **eigenvector** $\vec{v}$\vec{v}.

Role of Eigenvalues & Eigenvectors in Machine Learning

---

1. Principal Component Analysis (PCA)

- PCA finds the **eigenvectors** of the **covariance matrix** of the data.

- These eigenvectors (called **principal components**) represent **directions of maximum variance**.
- The corresponding **eigenvalues** tell **how much variance** each component captures.

We select top **k eigenvectors** (with highest eigenvalues) to **reduce dimensionality**.

---

2. Covariance and Correlation Analysis

- Eigenvalues help assess the **spread or variance** in each direction.
- Helps in identifying **redundant features**.

---

3. Spectral Clustering

- Uses **eigenvectors of similarity matrices** to reduce data before applying clustering.

---

4. Optimization & Stability

- In training models (like deep networks), eigenvalues of the **Hessian matrix** help analyze **curvature**, convergence speed, and **stability**.

---

5. Singular Value Decomposition (SVD)

- SVD is related to eigen decomposition of ATAA^T A and AATA A^T.
- Used in **image compression**, **recommendation systems**, and **matrix factorization**.

Q. Define and explain diagonalization of a matrix and method of diagonalising a matrix

A square matrix A is said to be **diagonalizable** if it can be written in the form:

$$A = PDP^{-1}$$

Where:

- A is the original **n×n** matrix
- P is a matrix whose **columns are eigenvectors** of A
- D is a **diagonal matrix** whose **diagonal entries are the eigenvalues** of A
- P^{-1} is the **inverse of matrix P**

Diagonalization means expressing a matrix in a form where **all off-diagonal entries are zero** — making computations like powers and exponentials much easier.

Why Diagonalize a Matrix?

Diagonal matrices are easy to:

- Multiply
- Invert
- Raise to a power

Method of Diagonalizing a Matrix:

Let's break it into steps:

Step 1: Find the Eigenvalues of A

Solve the **characteristic equation**:

$$\det(A - \lambda I) = 0$$

to get eigenvalues λ1,λ2,…,λn

Step 2: Find the Eigenvectors

For each eigenvalue λ, solve:

$$(A - \lambda I)\vec{v} = 0$$

This gives you **eigenvectors v⃗1,v⃗2,…**

Step 3: Form Matrix P

Let:

$$P = [\vec{v}_1\ \vec{v}_2\ \dots\ \vec{v}_n]$$

Each column is an eigenvector.

Step 4: Form Diagonal Matrix D

Let:

$$D = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

Each diagonal entry is the corresponding eigenvalue.

Step 5: Verify Diagonalization

Check if:

$$A = PDP^{-1} \quad \text{or equivalently} \quad D = P^{-1}AP$$

If so, matrix A is **diagonalizable**.

Conditions for Diagonalizability:

Condition	Meaning
A must have **n linearly independent eigenvectors**	⇒ Matrix is diagonalizable
If A is **symmetric**	⇒ always diagonalizable (very useful in ML)

Applications in Machine Learning:

- **PCA (Principal Component Analysis):** Diagonalizes covariance matrix to extract principal components.
- **Spectral clustering:** Uses eigen decomposition of graph Laplacians.
- **Efficient Computation:** Powers and exponentials of diagonal matrices are very fast to compute.

Q. Why are eigenvectors important in data transformation

In **data transformation**, especially in techniques like **Principal Component Analysis (PCA)** and **Singular Value Decomposition (SVD)**, **eigenvectors** play a **central role** in finding **new axes (directions)** along which the data has maximum variation.

---

1. Eigenvectors Identify Principal Directions

- Eigenvectors show the **directions** along which the data **varies the most**.
- When we transform data (e.g., using PCA), we are projecting it onto these **principal directions** to capture the **most informative components**.

---

2. Used in Dimensionality Reduction (e.g., PCA)

- In PCA, we compute the **eigenvectors of the covariance matrix** of the data.
- These eigenvectors define the **new axes** for the transformed data space.
- The top **k eigenvectors** (with highest eigenvalues) are selected to form a **lower-dimensional subspace**.

---

3. Provide a Basis for Transformation

- Eigenvectors **diagonalize** a matrix.
- This simplifies data transformation into a form that's **easier to interpret and compute**.
- The transformation aligns the axes with directions of **independent variation**.

---

4. Noise Reduction and Compression

- Low eigenvalue directions usually represent **noise** or **irrelevant features**.
- By removing low-eigenvalue eigenvectors, we **denoise** and **compress** the data.

---

5. Feature Decorrelation

- Transforming data using eigenvectors can **remove correlation** between features.
- This is helpful in models where uncorrelated features improve performance (e.g., Naive Bayes).

Q. Define trace, determinant

1. Trace of a Matrix

Definition: The **trace** of a **square matrix** is the **sum of all elements on its main diagonal** (from top-left to bottom-right).

Significance of Trace in ML:

- In PCA, the **trace of the covariance matrix** equals the **total variance** of the dataset.
- Sum of eigenvalues of a matrix is equal to its **trace**:
- Used in **matrix optimization**, loss functions, and **regularization**.

---

2. Determinant of a Matrix

Definition: The **determinant** is a scalar value that can be computed from a square matrix and gives information about:

- **Invertibility**
- **Volume scaling** under transformation
- **Linear dependence** of rows/columns

Significance of Determinant in ML:

- A matrix is **invertible only if** det(A)≠0\det(A) \neq 0
- In **linear transformations**, the determinant tells how **area or volume changes**
- In **multivariate probability distributions** (e.g., Gaussian), determinant of the covariance matrix is used in the **normalization constant**
- In PCA, determinant reflects **how much variance** is retained after projection

---

Properties:

Operation	Effect on Trace	Effect on Determinant
Transpose	Same	Same
Matrix multiplication	Additive for Tr	Multiplicative for det
Identity Matrix In	Tr = nn	det = 1
Singular Matrix	Tr is finite	det = 0 (not invertible)

Q. Explain SVD and mention its applications

Singular Value Decomposition (SVD) is a **matrix factorization technique** that decomposes any real (or complex) matrix A into three components:

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

Where:

- A is an **m × n** matrix
- U is an **m × m orthogonal matrix** (left singular vectors)
- Σ is an **m × n diagonal matrix** with non-negative **singular values**
- V^T is the **transpose** of an **n × n orthogonal matrix** (right singular vectors)

SVD works even if A is not square or not invertible.

Visual Breakdown of the Formula:

$$\text{Input Matrix } A = U \cdot \Sigma \cdot V^T$$

- **U** rotates data in input space
- **Σ** scales the data (via singular values)
- **V^t** rotates data in output space

Geometric Intuition:

Think of SVD as stretching, rotating, and flipping a vector in space.

- U rotates the coordinate system (basis in input space)
- Σ stretches (scales) the axes by singular values
- V^T rotates the vector in the output space

Applications of SVD in Machine Learning & Data Science:

---

1. Dimensionality Reduction (PCA)

- PCA is based on SVD of the **covariance matrix** of the data.
- We use top **k singular values** and their vectors to reduce dimensions while preserving **maximum variance**.

---

2. Noise Reduction

- In signal and image processing, small singular values (representing noise) can be **removed**, leaving only important structure.

Used in denoising MRI scans, satellite images, etc.

---

3. Image Compression

- Convert an image to a matrix and apply SVD.
- Keep only top **k singular values** and corresponding vectors — this reduces **storage size** with minimal loss of quality.

---

4. Recommender Systems (e.g., Netflix, Amazon)

- User-item rating matrix is factorized using SVD.
- Predicts missing entries (e.g., what movie a user might like).

---

5. Natural Language Processing (NLP)

- Used in **Latent Semantic Analysis (LSA)** to capture relationships between words and documents.

---

6. Solving Linear Systems & Pseudo-Inverses

- If a matrix is **non-invertible or rectangular**, SVD helps compute its **Moore-Penrose pseudo-inverse** to solve equations.

Q.