Projection matrix explanation

explain projection matrix

Alright Vernika, let's go step-by-step and build full intuition for **Projection Matrices** — what they are, why they're useful in machine learning, and how they look mathematically.

---

1. What is a Projection Matrix?

A **projection matrix** transforms a vector into its **projection** on a given subspace.

- Think of it as **dropping a shadow** of a vector onto a plane or line.
- Once a vector is projected, applying the same projection again doesn't change it — because it's **already in the subspace**.

---

2. Core Properties

If **P** is a projection matrix:

1. **Idempotent**:
   $$P^2 = P$$
   (Projecting twice is the same as projecting once)
2. **Symmetric** (for orthogonal projections):
   $$P^T = P$$
3. **Eigenvalues**:
   Only **0 or 1** —
   - Eigenvalue 1 → directions in the subspace (stay the same after projection)
   - Eigenvalue 0 → directions orthogonal to the subspace (collapse to zero)

---

3. Visual Intuition

Imagine:

- **A line** in 2D space defined by a vector **u**
- A point **v** anywhere in the plane
- The projection matrix takes **v** and "drops" it onto the line so it lies perfectly in the same direction as **u**.

Diagram (conceptual)

```pgsql
v ●
  \
   \
    \
     ● v_proj   → lies on the line spanned by u
    /
   /
-----●--------------- line spanned by u
```

---

4. Formula for Projection onto a Vector

If we have a vector u (not zero), the projection matrix onto the space spanned by u is:

$$P = \frac{uu^T}{u^T u}$$

So, for any vector v:

$$\text{proj}_u(v) = Pv$$

Why this formula works:

- $u^T v \rightarrow$ gives the amount of v along u (dot product)
- Dividing by $u^T u \rightarrow$ normalizes u
- Multiplying by u again → turns that scalar back into a vector along u

---

5. Projection onto a Subspace

If we have a matrix A whose columns are **basis vectors of the subspace**:

Projection matrix is:

$$P = A(A^T A)^{-1} A^T$$

- **A**: $m \times n$ matrix (n basis vectors in m-dimensional space)
- $(A^T A)^{-1}$ → adjusts for basis not being orthonormal

- This formula works even when the basis vectors are not orthogonal.

---

6. Example (Numerical)

Project $v = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ onto the line defined by $u = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

Step 1: Find P

$$P = \frac{uu^T}{u^T u} = \frac{\begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix}}{1^2 + 2^2} = \frac{\begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}}{5}$$

$$P = \begin{bmatrix} 0.2 & 0.4 \\ 0.4 & 0.8 \end{bmatrix}$$

Step 2: Apply P to v

$$Pv = \begin{bmatrix} 0.2 & 0.4 \\ 0.4 & 0.8 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2.2 \\ 4.4 \end{bmatrix}$$

So, projection of v onto the line is $\begin{bmatrix} 2.2 \\ 4.4 \end{bmatrix}$.

---

7. Why Projection Matrices Matter in Machine Learning

- **Linear Regression**: The fitted values are projections of y onto the column space of X:

  $$\hat{y} = Py, \quad P = X(X^T X)^{-1} X^T$$

  This is why we call P the **hat matrix** (it puts the hat on y).
- **Dimensionality Reduction (PCA)**: Projecting data onto top k principal components.
- **Computer Graphics**: Projection from 3D world to 2D screen.
- **Signal Processing**: Removing components orthogonal to the signal of interest.

If you want, I can make **a detailed Word-compatible LaTeX + visual sheet** where I show:

- Projection on a vector

- Projection on a plane
- Orthogonal vs oblique projection
- The connection to linear regression's hat matrix

Do you want me to prepare that visual + LaTeX version next?

can you explain subspace and intutively what is a projection matrix and why we need it. its advanatges disadvantages and where used in Ml and also visual to support

Got it — let's go step-by-step, starting with **what a subspace is**, then building the **intuition** for a projection matrix, and finally connecting it to **ML use cases, advantages, disadvantages**, and a **visual**.

---

1. What is a Subspace? (Intuition First)

A **subspace** is simply a "flat surface" that passes through the origin in a higher-dimensional space.

Think of:

- A **line through the origin** in 2D → 1D subspace of 2D space.
- A **plane through the origin** in 3D → 2D subspace of 3D space.
- A **3D volume through the origin** in 5D → 3D subspace of 5D space.

Mathematically:

- It must be **closed under addition** (sum of two vectors in the space stays in the space)
- It must be **closed under scalar multiplication** (scaling a vector still stays in the space)
- It **always contains the zero vector**.

Subspace Example

If $u = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

Then all multiples of u form a 1D subspace of R^2:

$$\text{Span}(u) = \{\alpha u \mid \alpha \in \mathbb{R}\}$$

---

2. Intuition: What is a Projection Matrix?

Imagine you're standing under the sun — your **shadow** on the ground is a **projection** of you onto the ground plane.

A **projection matrix** is just the mathematical way to:

- Take a vector in **the full space**,
- **Drop it** onto a subspace,
- And get the "closest point" in that subspace to the original vector.

💡 **Key Intuition**:
Projection is about **minimizing distance** from the original vector to the subspace.

How It Works

If:

- **Subspace** = "target surface" (line, plane, etc.)
- **Projection** = "closest point on that surface to your vector"
- **Projection matrix** = "machine that does this automatically"

---

3. Why Do We Need Projection Matrices?

In many problems, data doesn't lie in the full high-dimensional space — it actually sits close to a smaller subspace.

We project to:

1. **Reduce dimensions** (e.g., PCA)
2. **Remove noise** (keep only the components in the desired subspace)
3. **Find the best approximation** in that subspace

---

4. Advantages & Disadvantages

✅ Advantages

- **Dimensionality reduction** → lower computational cost
- **Noise filtering** → ignore directions irrelevant to the problem
- **Mathematical clarity** → makes "best fit" problems easy (like linear regression)

⚠ Disadvantages

- **Information loss** → components orthogonal to the subspace are gone forever
- **Choice of subspace matters** → if the chosen subspace doesn't capture the important features, results suffer
- **Computational cost** → large projection matrices can be expensive for huge datasets

---

5. Where It's Used in Machine Learning

ML Area	Use of Projection Matrix
Linear Regression	"Hat matrix" $P = X(X^TX)^{-1}X^T$ projects y into the column space of X
PCA	Projects data onto top k principal components
Least Squares	Projection finds the closest solution in subspace spanned by features
Feature Extraction	Projects raw data into new basis for better learning
Computer Vision	Projection from 3D world to 2D image plane
Signal Processing	Removes unwanted components by projecting onto desired frequency subspace

---

6. Visual Intuition

Here's the geometry for a vector v being projected onto a subspace (line):

```arduino
  v ●
    \
     \
      \
       ● v_proj    (lies in subspace)
      /
     /
-----●-----------------  Subspace (line or plane)
```

In higher dimensions, the idea is the same — we just "drop" the vector to the nearest point in the subspace.

---

7. Mathematical Form (for Reference)

- Projection onto a single vector u:

$$P = \frac{uu^T}{u^T u}$$

- Projection onto a subspace spanned by columns of A:

$$P = A(A^T A)^{-1} A^T$$

Where A has basis vectors as its columns.

If you want, I can now **create a one-page Word-compatible LaTeX + diagrams sheet** summarizing:

- Subspace definition & examples
- Projection intuition
- Formula
- ML use cases
- Visual diagrams for line and plane projection

That way, you can directly paste it into your notes.
Do you want me to make that next?

so basically subsapce is a space that is part of full space. lets say line is a subsapce of a 2D space and plane is a subsapce of 3D spcae

Exactly ✅

That's the core idea —

- In **2D space** (R^2):
  - A **line through the origin** is a **1D subspace**.
  - The entire plane itself is the **2D subspace**.
- In **3D space** (R^3):
  - A **line through the origin** is a **1D subspace**.
  - A **plane through the origin** is a **2D subspace**.
  - The whole 3D space is the **3D subspace**.

💡 **Rule:** Subspaces must always pass through the origin and be "closed" under vector addition and scalar multiplication — meaning:

- Adding two vectors from the subspace keeps you in the subspace
- Scaling any vector in the subspace keeps you in the subspace

This is why a tilted line **not passing through the origin** is **not** a subspace — it's just an affine subset.

If you want, I can also make a **visual comparison of a subspace vs a non-subspace** so it's crystal clear.
Do you want me to make that visual now?

yes

Not A Subspace

Output image

Here's the visual:

- **Left:** A valid subspace → line passes through the origin and is closed under addition and scalar multiplication.
- **Right:** Not a subspace → line is shifted away from the origin, so adding vectors or scaling them can take you outside the line.

projection matrix is a matrix that reduce the dimension of data by projecting data into lower subspace. It does by finding the points nearer to original point or vector in subspace. Lets say original space is 2D and I create a subsapce line passing throgh origin. then if I bring all data on that line then it is projection

Yes, that's the right idea — you've basically described the **geometric meaning** of a projection matrix.

Let's refine it a bit so it's precise both **intuitively** and **mathematically**:

Refined Understanding

- **What it does**:
  A **projection matrix** takes a vector from a higher-dimensional space and maps it to its **closest point** in a given subspace (like a line or plane through the origin).
- **Dimension reduction**:
  If the subspace has **fewer dimensions** than the original space, then yes — projection reduces dimensionality.
  (Though in matrix form, the vector stays in the same coordinate space; it just *lies within* the lower-dimensional subspace.)

- **Example**:

  If you have:

  - Original space: R^2
  - Subspace: 1D line through the origin
  - You "drop" every point in R^2 onto that line

    → That's projection.

Key Geometric Idea

It finds:

$$\hat{v} = \arg \min_{w \in \text{Subspace}} \| v - w \|$$

Meaning:

"Find the point $\hat{v}$ in the subspace that is **closest** to the original vector v."

Why We Care in ML

- This "closest point" property makes projection matrices the core of:
  - **Linear regression** (projecting y onto the space spanned by X)
  - **PCA** (projecting onto principal component directions)
  - **Noise reduction** (remove components orthogonal to the signal)

If you like, I can now **draw the exact diagram** for your example:

Original space = 2D,

Subspace = a line,

Projection = points "fall" onto the line,

and show the **before-and-after transformation**.

Do you want me to prepare that?

yes

Projection From 2D To 1D Subspace (Line Through Origin)

Output image

Here’s the diagram:

- **Red dots** = Original data points in 2D space.
- **Green line** = 1D subspace (line through origin).
- **Blue dots** = Projected points lying on the subspace.
- **Gray dashed lines** = Shortest path from each original point to its projection (minimizing distance).

This is exactly how a **projection matrix** works in your example.

can you show noise reduction example also

Noise Reduction Using Projection Matrix

Output image

Here's the **noise reduction example**:

- **Green line** → True underlying signal (subspace).
- **Red dots** → Noisy observations scattered around the signal.
- **Blue dots** → Data after projection onto the signal's subspace — noise removed, only the true component remains.
- **Gray dashed lines** → How each noisy point "falls" onto the subspace during projection.

This is exactly how projection helps in **ML** to filter out components orthogonal to the signal (noise).

so here information loss is there and I think we can't undo it.

Exactly right! 🔥

When you project noisy data onto a lower-dimensional subspace (like the line in the example), you are:

- **Removing components orthogonal to the subspace** (which includes noise).
- This means **some information is lost** — specifically, anything that was in the directions *outside* that subspace.

Why you can't undo it:

- Projection is **not invertible** if you reduce dimension (going from 2D to 1D in the example).
- After projection, you only have the "shadow" in the subspace — the exact original point cannot be recovered because many different points map to the same projection.
- So the **information in orthogonal directions is lost permanently**.

Trade-off in ML:

- **Advantage**: Noise is reduced → cleaner signal, better model generalization.
- **Disadvantage**: Potentially useful signal in orthogonal directions might also be lost, causing bias or underfitting.

That's why choosing the right subspace (e.g., top principal components in PCA) is crucial — to **preserve maximum useful information while discarding noise**.

If you want, I can write this explanation with a Word-compatible LaTeX snippet summarizing these pros, cons, and the irreversibility of projection. Would that help?

can you explain that part where you said, points remain in the same space or something similar

Sure! Let me clarify that part about "points remain in the same space" when you apply a projection matrix, especially in the context of dimensionality reduction.

What I meant:

When you **apply a projection matrix P to a vector v in** $\mathbb{R}^n$, the **result Pv is still an n-dimensional vector**.

- So the projected vector **lives in the same ambient space** $\mathbb{R}^n$ as the original.
- However, the key is that Pv **lies entirely inside the lower-dimensional *subspace* within** $\mathbb{R}^n$.

Example:

- Original space: $\mathbb{R}^2$ (2D vectors)
- Subspace: A line through the origin (1D) in $\mathbb{R}^2$

Applying projection P to any vector $v \in \mathbb{R}^2$:

- $Pv \in \mathbb{R}^2$ still — same dimension as v
- But now, Pv is **on the line (subspace)** inside $\mathbb{R}^2$, meaning it can be written as αu for some scalar α, where u is the basis vector of the line.

Why this matters:

- The dimension of the vector space hasn't changed: input and output vectors have the same number of components.
- But the **output vectors lie in a smaller "flat" subset (subspace)** of that space — so effectively, you have *reduced the degrees of freedom* even though the vector length is the same.

In contrast:

- If you **explicitly reduce dimension** (e.g., project and then drop coordinates), you get vectors with fewer components (e.g., from 2D → 1D vector like just the scalar α).
- But **the projection matrix itself**, as a linear operator, maps vectors to vectors in the *same* ambient space, just constrained to the subspace.

Summary in short Word-compatible LaTeX:

$$P : \mathbb{R}^n \to \mathbb{R}^n, \quad \text{with} \quad P^2 = P, \quad \text{and} \quad \text{Range}(P) = \text{subspace} \subseteq \mathbb{R}^n$$

$$\forall v \in \mathbb{R}^n, \quad Pv \in \text{subspace} \subseteq \mathbb{R}^n$$

Let me know if you want me to create a visual or a document snippet for this explanation!

yes

Here's a clean visual + explanation you can use for your notes, showing that the projection matrix maps vectors in $\mathbb{R}^n$ back into $\mathbb{R}^n$ but constrained inside the subspace:

Visual Explanation

```cpp
Original vector v in R^n
       ●
       |
       |    \
       |     \  (orthogonal component)
       |      \
-------●--------●--------- Subspace (lower dim) in R^n
       |      /
       |     /
       |    /  (projection Pv lies here)
       ● Pv
```

Word-compatible LaTeX Explanation

```pgsql
P : \mathbb{R}^n \to \mathbb{R}^n,\quad
P^2 = P, \quad
P^T = P \quad \text{(for orthogonal projection)}

\text{Range}(P) = \text{subspace} \subseteq \mathbb{R}^n

\text{For any } v \in \mathbb{R}^n, \quad Pv \in \text{subspace} \subseteq \mathbb{R}^n
```

Summary

- The vector v and the projected vector Pv both live in the same n-dimensional space.
- But Pv lies in a **lower-dimensional subspace** inside that space (e.g., a line or plane).
- The projection matrix **does not reduce the dimension of the output vector itself**, but restricts it to the subspace.

If you want, I can prepare this with a clean diagram file or a Word doc snippet next. Would you like that?

you mean to say no of rows and column reamin same

Exactly!

When you apply a **projection matrix** P to a vector v:

- P is always a **square matrix** of size $n \times n$ (same number of rows and columns).
- The input vector v is an $n \times 1$ column vector.
- The output vector Pv is also an $n \times 1$ column vector — so **dimension of the vector doesn't change**.

But:

- The **output vector lies in the subspace**, which is a smaller-dimensional "flat" within the n-dimensional space.
- So even though Pv has n components, it only varies along fewer directions (the subspace basis vectors).

So yes, matrix size stays the same (square $n \times n$) and vector length stays the same, but the effective dimension of the output is smaller because it lives in a subspace.

If you want, I can write this as a short Word-compatible LaTeX snippet too!

so either some values get zero or get a combination of other points ?

Exactly! You nailed it. Here's the detailed intuition:

When you apply the projection matrix P to a vector v:

- The output vector Pv is **a linear combination of the basis vectors of the subspace**.
- This means:
  - Some components (directions orthogonal to the subspace) effectively become **zero** after projection.
  - The rest are **re-expressed as combinations** of the subspace basis vectors.

To put simply:

- If your original vector v had components **both inside and outside** the subspace,
- The projection **keeps only the part inside the subspace** and **removes (zeroes out) the rest**.

Visual analogy:

Think of a shadow of a 3D object on the floor:

- The parts "above" the floor vanish in the shadow (zeroed out).
- The parts on the floor plane combine to form the shadow shape (linear combination).

Word-compatible LaTeX snippet summarizing this:

$$Pv = \sum_{i=1}^{k} \alpha_i u_i, \quad u_i \text{ are basis vectors of subspace,} \quad \alpha_i \in \mathbb{R}$$

$$\text{and components orthogonal to } \{u_i\} \text{ become zero.}$$

Let me know if you want me to prepare a diagram or detailed notes for this!

can you give a diagram of showing which part gets zero or how lying in same subsapce

Sure! Here's a simple diagram illustrating a 3D vector projected onto a 2D plane subspace, showing which component gets zeroed out (the orthogonal part) and how the projected vector lies in the subspace:

---

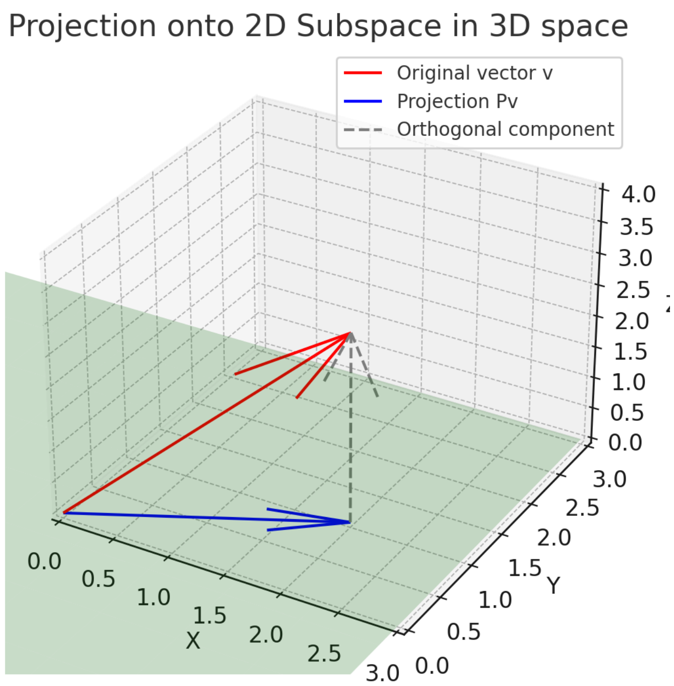
perl

```
          v (original vector)
          /
         /
        /|
       / |
      /  |
     /   | orthogonal component (gets zeroed)
    /    |
   ●-----●------------ Subspace (2D plane)
  Pv (projection)
```

- The original vector v has two parts:
  - **One inside the subspace (plane)** — this is Pv (blue dot).
  - **One orthogonal to the subspace** — this part gets "zeroed out" by projection.
- The projection matrix keeps only the subspace part and removes the orthogonal component.

I'll also generate a plotted image with this visual in 3D for you. Just a moment.

Projection Onto 2D Subspace In 3D Space



Here's the 3D visual:

- **Red vector** v = original vector in 3D space.
- **Blue vector** Pv = projection of v onto the 2D plane subspace (green translucent plane).
- **Gray dashed vector** = orthogonal component $v - Pv$, which gets zeroed out by projection.

This shows how the projection keeps only the part lying in the subspace and removes the perpendicular part.

can you give me example of projection matrix

Sure! Here's a concrete example of a **projection matrix** projecting vectors from R^2 onto a line through the origin.

Example: Projection onto the line spanned by vector $u = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$

Step 1: Compute the projection matrix P

$$P = \frac{uu^T}{u^T u}$$

Calculate denominator:

$$u^T u = 3^2 + 4^2 = 9 + 16 = 25$$

Calculate numerator:

$$uu^T = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \begin{bmatrix} 3 & 4 \end{bmatrix} = \begin{bmatrix} 9 & 12 \\ 12 & 16 \end{bmatrix}$$

So,

$$P = \frac{1}{25} \begin{bmatrix} 9 & 12 \\ 12 & 16 \end{bmatrix} = \begin{bmatrix} 0.36 & 0.48 \\ 0.48 & 0.64 \end{bmatrix}$$

Step 2: Use P to project a vector v

Let's project $v = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$:

$$Pv = \begin{bmatrix} 0.36 & 0.48 \\ 0.48 & 0.64 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 1.8 \\ 2.4 \end{bmatrix}$$

Interpretation:

- Original vector $v = (5, 0)$ is projected onto the line spanned by $u = (3, 4)$.
- The projected vector $Pv = (1.8, 2.4)$ lies **on the line** through u.
- Notice $Pv = 0.6 \times u$ (since $0.6 \times 3 = 1.8, 0.6 \times 4 = 2.4$).

If you want, I can prepare this as a Word-compatible LaTeX snippet for your notes. Would you like that?