UNIT – 4 : LINEAR ALGEBRA

Basics:

Linear algebra in data science refers to the use of mathematical concepts involving vectors, matrices, and linear transformations to manipulate and analyze data. It provides useful tools for most algorithms and processes in data science, such as machine learning, statistics, and big data analytics.

In the field of data science, linear algebra supports various tasks. These include algorithm design, data processing, and machine learning. With linear algebra, complex problems become simpler. It turns theoretical data models into practical solutions that can be applied in real-world situations.

Importance of Linear Algebra in Data Science:

Understanding linear algebra is key to becoming a skilled data scientist. Linear algebra is important in data science because of the following reasons:

- It helps in organizing and manipulating large data sets with efficiency.
- Many data science algorithms rely on linear algebra to work fast and accurately.
- It supports major machine learning techniques, like regression and classification.
- Techniques like Principal Component Analysis for reducing data dimensionality depend on it.
- Linear algebra is used to alter and analyze images and signals.
- It solves optimization problems, helping find the best solutions in complex data scenarios.

Key Concepts in Linear Algebra:

Linear algebra is a branch of mathematics useful for understanding and working with arrays of numbers known as matrices and vectors. Let us understand some of the key concepts in linear algebra in the table below:

Concept	Description
Vectors	Fundamental entities in linear algebra representing quantities with both magnitude and direction, used extensively to model data in data science.
Matrices	Rectangular arrays of numbers, which are essential for representing and manipulating data sets.

Matrix Operations	Operations such as addition, subtraction, multiplication, and inversion that are crucial for various data transformations and algorithms.
Eigenvalues and Eigenvectors	These are used to understand data distributions and are crucial in methods such as Principal Component Analysis (PCA) which reduces dimensionality.
Singular Value Decomposition (SVD)	A method for decomposing a matrix into singular values and vectors, useful for noise reduction and data compression in data science.
Principal Component Analysis (PCA)	A statistical technique that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables.

Challenges in Learning Linear Algebra for Data Science:

Let us learn about some of the most common challenges in learning linear algebra for data science -

Abstract Concepts: Linear algebra involves many abstract concepts like vectors, matrices, and transformations. These can be hard to visualize. For beginners, understanding how these concepts translate to solving real-world data problems is often challenging. A common struggle is seeing how theoretical matrix operations apply to practical tasks like image recognition.

Steep Learning Curve: The learning curve for linear algebra is steep, especially for those without a strong mathematical background. Learning to perform operations like matrix inversion and eigenvalue decomposition can be daunting. For instance, mastering eigenvalues and eigenvectors is crucial for PCA, but understanding their importance and computations takes some effort.

Bridging Theory and Practice: Applying linear algebra in data science requires bridging theory with practical application. Learners often find it difficult to connect the dots between abstract mathematical theories and their practical implementation in software like Python's NumPy or MATLAB. This gap makes it hard to apply learned concepts directly to data science projects.

Overwhelming Range of Applications: Linear algebra is used in a wide range of data science applications, from natural language processing to computer vision. For learners, understanding where to apply specific linear algebra techniques across different domains can be overwhelming. Each field may use the same mathematical tools in subtly different ways.

Matrices to Represent Relations Between Data:

In data science, matrices are fundamental structures for representing and manipulating relationships between data points.

Below are some common types of matrices and their uses in various data science tasks:

1. Adjacency Matrix:

An adjacency matrix is used to represent graphs. It is a square matrix used to represent a finite graph, with the element at row *ii* and column *jj* representing the presence (and possibly the weight) of an edge between vertices *ii* and *jj*.

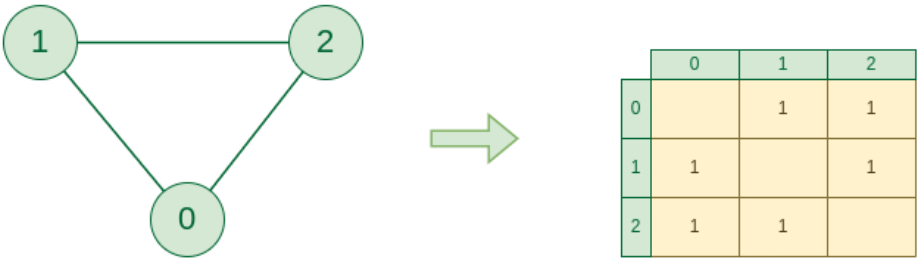
Characteristics of the adjacency matrix are:

- The size of the matrix is determined by the number of vertices in the graph.
- The number of nodes in the graph determines the size of the matrix.
- The number of edges in the graph is simply calculated.
- If the graph has few edges, the matrix will be sparse.

How to build an Adjacency Matrix:

- Create an **n x n** matrix where **n** is the number of vertices in the graph.
- Initialize all elements to 0.
- For each edge (u, v) in the graph, if the graph is undirected mark a[u][v] and a[v][u] as 1, and if the edge is directed from **u** to **v**, mark a[u][v] as the 1. (Cells are filled with edge weight if the graph is weighted)

Example:



Undirected Graph Adjacency Matrix

- **Example Use**: Social networks, where nodes represent users and edges represent friendships or interactions.

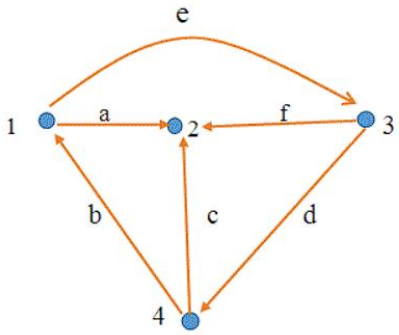
2. Incidence Matrix:

An incidence matrix is another way to represent a graph. It shows the relationship between vertices and edges. Each row corresponds to a vertex, and each column corresponds to an edge. This matrix can be denoted as $[A_C]$ As in every matrix, there are also rows and columns in **incidence matrix** $[A_C]$.

The rows of the matrix $[A_C]$ represent the number of nodes and the column of the matrix $[A_C]$ represent the number of branches in the given graph. If there are 'n' number of rows in a given incidence matrix, that means in a graph there are 'n' number of nodes. Similarly, if there are 'm' number of columns in that given incidence matrix, that means in that graph there are 'm' number of branches.

Type of branch	**Value**
Outgoing branch from k^{th} node	+1
Incoming branch to k^{th} node	-1
Others	0

Example:



For the graph shown above write its incidence matrix.

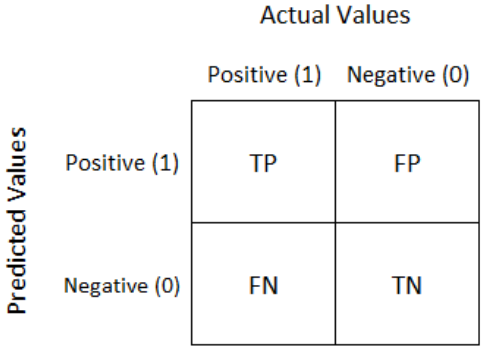
$[A_C]$ =

nodes \ branches	a	b	c	d	e	f
1	1	-1	0	0	1	0
2	-1	0	-1	0	0	-1
3	0	0	0	1	-1	1
4	0	1	1	-1	0	0

- **Example Use**: Network analysis, where the relationships between nodes (vertices) and their connections (edges) need to be explicitly represented.

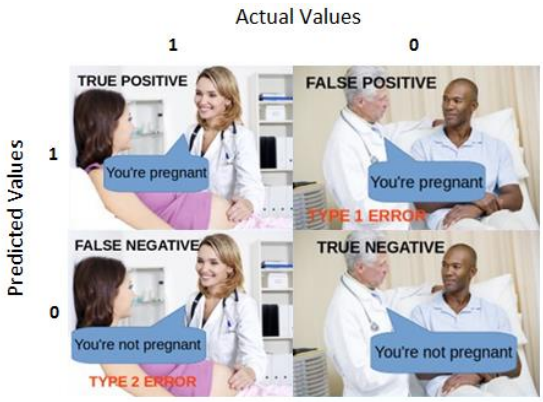
3. Confusion Matrix:

A confusion matrix is used to evaluate the performance of a classification algorithm. It shows the actual versus predicted classifications.



It is extremely useful for measuring Recall, Precision, Specificity, Accuracy, and most importantly AUC-ROC curves.

Let's understand TP, FP, FN, TN in terms of pregnancy analogy.



True Positive: You predicted positive and it's true.

True Negative: You predicted negative and it's true.

False Positive (Type 1 Error): You predicted positive and it's false.

False Negative (Type 2 Error): You predicted negative and it's false.

Recall: The below equation can be explained by saying, from all the positive classes, how many we predicted correctly. Recall should be high as possible.

$$Recall = \frac{TP}{TP + FN}$$

Precision: The below equation can be explained by saying, from all the classes we have predicted as positive, how many are actually positive. Precision should be high as possible.

$$Precision = \frac{TP}{TP + FP}$$

Accuracy: From all the classes (positive and negative), how many of them we have predicted correctly. Accuracy should be high as possible.

$$Accuracy = \frac{TP + TN}{Total}$$

F-measure: It is difficult to compare two models with low precision and high recall or vice versa. So, to make them comparable, we use F-Score.

F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

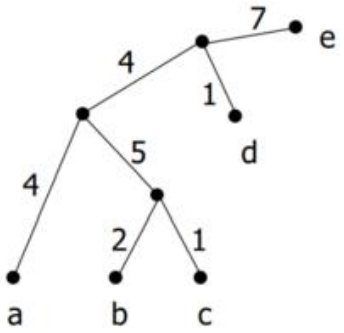
$$F\text{-}measure = \frac{2*Recall*Precision}{Recall + Precision}$$

4. Distance Matrix:

A distance matrix stores the distances between pairs of objects. It is typically used in clustering algorithms and other methods that require a notion of distance or similarity. The distance matrix is a symmetric quadratic matrix of size n × n which contains all zeroes along the main diagonal (the distance of each object and a replica of itself is zero).

Example:

M	a	b	c	d	e
a	0	11	10	9	15
b	11	0	3	12	18
c	10	3	0	11	17
d	9	12	11	0	8
e	15	18	17	8	0



- **Example Use**: K-means clustering, where the distances between data points need to be calculated to form clusters.

5. Correlation Matrix:

A correlation matrix shows the pairwise correlation coefficients between variables. It is used to identify relationships between variables.

- **Example Use**: Feature selection in machine learning, to find which features are highly correlated.

How does the correlation matrix work?

The correlation matrix calculates the linear relationship between two variables. The matrix is constructed by computing the correlation coefficient for each pair of variables and inserting it into the relevant cell of the matrix.

The following formula is used to compute the correlation coefficient between two variables:

$$r = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{\sqrt{[n\Sigma x^2 - (\Sigma x)^2][n\Sigma y^2 - (\Sigma y)^2]}}$$

where:

r = correlation coefficient

n = number of observations

The resulting correlation coefficient varies from -1 to +1, with -1 being a perfect negative correlation, +1 representing a perfect positive correlation, and 0 representing no correlation between the variables.

Example:

Let's look at an example to see how a correlation matrix can help people read and understand a dataset with four variables: age, income, education, and job satisfaction:

	Age	Income	Education	Job Satisfaction
Age	1	0.5	0.3	0.2
Income	0.5	1	0.8	0.6
Education	0.3	0.8	1	0.4
Job Satisfaction	0.2	0.8	0.4	1

In this example, we can see that income and education have a strong positive correlation of 0.8. This means that people with higher education levels tend to have higher incomes. Age and income also have a moderately positive correlation of 0.5, suggesting that income increases as people age. But the correlation between age and job satisfaction is only 0.2, which shows that age is not a strong predictor of job satisfaction.

6. Design Matrix (X Matrix):

In regression analysis, the design matrix (or model matrix) contains the values of the explanatory variables (predictors) for all observations.

Example: If we collect the data about the gross domestic product (GDP) of four countries in three consecutive years, then the design matrix is the 4*3 matrix

$$X = \begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \\ X_{31} & X_{32} & X_{33} \\ X_{41} & X_{42} & X_{43} \end{bmatrix}$$

where, for example, X_{32} is the GDP of the third country in the second year.

- **Example Use**: Linear regression, where each row represents an observation and each column represents a predictor variable.

7. Transition Matrix:

A transition matrix represents the probabilities of transitioning from one state to another in a stochastic process.

Single Step Transitions: In the simplest possible setup, the stochastic system transitions from an initial state to a final state in a single step.

Let's assume a state space $S = 0, \ldots, D$. If the Transition Probability (the probability of moving from state m to state n) in one time step is $Pr(n|m) = T^{mn}$, the transition matrix T is given by using T^{mn} as the m^{th} row and n^{th} column element:

$$P = \begin{pmatrix} T^{00} & T^{01} & \ldots & T^{0n} & \ldots & T^{0D} \\ T^{10} & T^{11} & \ldots & T^{1n} & \ldots & T^{1D} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ T^{m0} & T^{m1} & \ldots & T^{mn} & \ldots & T^{mD} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ T^{D0} & T^{D1} & \ldots & T^{Dn} & \ldots & T^{DD} \end{pmatrix}.$$

Properties:

- Since the matrix elements are probabilities, they must satisfy

$$0 \leq T^{mn} \leq 1$$

- Since a system must migrate to any of the available states (inclusive of possibly staying in the same state) the transition matrix must satisfy:

$$\sum_{n=0}^{D} T^{mn} = 1.$$

Generator of a Transition Matrix:

The infinitesimal generator Q of a transition matrix T satisfies:

$$T = \exp(tQ) = I + tQ + \frac{(tQ)^2}{2!} + \frac{(tQ)^3}{3!} + \dots$$

- **Example Use**: Markov chains, which are used in various fields such as finance and natural language processing.

8. Rating Matrix:

A rating matrix is used in recommendation systems, where rows represent users and columns represent items, and the entries represent the ratings given by users to items.

RISK MATRIX						
PROBABILITY →	Very Likely - 5	4	10	15	20	25
	Likely - 4	4	8	12	16	20
	Possible - 3	3	6	9	12	15
	Unlikely - 2	2	4	6	8	10
	Very Likely - 1	1	2	3	4	5
		1	2	3	4	5
		Negligible	Slightly	Moderate	High	Very High
		SEVERITY →				

Risk	Risk level	Condition
1 to 6	Low Risk	*May be acceptable but review task to see if risk can be reduced further.*
8 to 12	Medium Risk	*Task should only be executing with appropriate management authorization after consulting with specialist personal.*
15 to 25	High Risk	*Task must not proceed, until adequate action taken to minimize the risk.*

- **Example Use**: Collaborative filtering in recommendation systems.

9. Feature Matrix: In machine learning and statistical modeling, a feature matrix represents the independent variables or features of the dataset. Each row in the matrix corresponds to an observation or instance, and each column represents a feature or attribute. This matrix is often used as input for training and evaluating models.

10. Similarity Matrix: A similarity matrix is used to represent the pairwise similarity or dissimilarity between entities or objects. Each element in the matrix represents the similarity score between two entities, such as documents, images, or data points. Similarity matrices are widely used in clustering, recommendation systems, and information retrieval tasks.

Linear Algebraic Operations on Matrices:

Matrix Addition: Two matrices can be added if they have the same dimensions. The result is a matrix where each element is the sum of the corresponding elements of the input matrices.

$$\mathbf{C} = \mathbf{A} + \mathbf{B}\text{ , where } C_{ij} = A_{ij} + B_{ij}$$

Matrix Subtraction: Similar to addition, two matrices can be subtracted if they have the same dimensions. The result is a matrix where each element is the difference of the corresponding elements of the input matrices.

$$\mathbf{C} = \mathbf{A} - \mathbf{B}\text{ , where } C_{ij} = A_{ij} - B_{ij}$$

Scalar Multiplication: A matrix can be multiplied by a scalar (a single number). Each element of the matrix is multiplied by the scalar.

$$\mathbf{B} = K\mathbf{A}\text{ , where } B_{ij} = k \cdot A_{ij}$$

Matrix Multiplication: The product of two matrices **A** and **B** is defined if the number of columns in **A** is equal to the number of rows in **B**. The result is a matrix **C** where each element is computed as the dot product of the corresponding row of **A** and column of **B**.

$$\mathbf{C} = \mathbf{AB}\text{ , where } C_{ij} = \sum A_{ik} B_{kj}$$

Matrix Transposition: The transpose of a matrix **A** is a new matrix $\mathbf{A}^T$ where the rows of **A** become the columns of $\mathbf{A}^T$ and vice versa.

$$(\mathbf{A}^T)_{ij} = A_{ji}$$

Determinant: The determinant is a scalar value that is a function of a square matrix. It provides important properties of the matrix and is used in solving linear systems, among other applications.

For a 2x2 matrix:

$$\det(\mathbf{A}) = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

For larger matrices, the determinant is computed using more complex methods such as Laplace expansion or row reduction.

Inverse: The inverse of a square matrix **A**, denoted $\mathbf{A}^{-1}$, is a matrix such that when multiplied by **A** results in the identity matrix. Not all matrices have inverses; a matrix has an inverse only if its determinant is non-zero.

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$$

Eigenvalues and Eigenvectors: For a given square matrix **A**, an eigenvector $\boldsymbol{v}$ and eigenvalue $\boldsymbol{\lambda}$ satisfy the equation:

$$\boldsymbol{Av} = \boldsymbol{\lambda v}$$

Eigenvalues and eigenvectors are important in many applications such as principal component analysis (PCA) and stability analysis.

Matrix Decomposition:

Matrix decomposition is a method of reducing or factoring a matrix into a set of product matrices. Another name for this is ***matrix factorization***. Working with these product matrices often makes it easier to carry out more complex matrix operations (e.g., computing an inverse).

In many ways, matrix decomposition is similar to factoring scalars. For example, we can factor the scalar 36 as:

$$36 = 12 \times 3$$

We could also have used the following factorizations:

$$36 = 9 \times 4$$

$$36 = 18 \times 2$$

$$36 = 6 \times 3 \times 2$$

$$36 = 2 \times 2 \times 3 \times 3$$

There are potentially multiple ways to factor a scalar, and in different applications, some of these factorizations may prove more useful than others. For example, the last factorization in the example is referred to as the ***prime factorization*** (as the factors of 36 are all prime numbers) and is useful for some mathematical applications.

The same is true of matrix decompoosition.

- There are many methods of matrix decomposition.
- Depending on the application, some of these methods are more useful than others.

Singular Value Decomposition (SVD):

Singular Value Decomposition (SVD) is a powerful matrix factorization technique used in many data science applications, including dimensionality reduction, noise reduction, and data compression. SVD decomposes a matrix into three constituent matrices, providing insights into the properties of the original matrix.

Mathematics behind SVD:

The SVD of m*n matrix A is given by the formula $A = U\Sigma V^T$

where:

- **U:** *m***m* matrix of the orthonormal eigenvectors of AA^T.
- **V^T:** transpose of a *n***n* matrix containing the orthonormal eigenvectors of $A^T A$.
- Σ: diagonal matrix with r elements equal to the root of the positive eigenvalues of AA^T or $A^T A$ (both matrics have the same positive eigenvalues anyway).

Example:

Find the singular value decomposition of a matrix

$$A = \begin{bmatrix} -4 & -7 \\ 1 & 4 \end{bmatrix}$$

Solution:

Given,

$$A = \begin{bmatrix} -4 & -7 \\ 1 & 4 \end{bmatrix}$$

So,

$$A^T = \begin{bmatrix} -4 & 1 \\ -7 & 4 \end{bmatrix}$$

Now,

$$AA^T = \begin{bmatrix} -4 & -7 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} -4 & 1 \\ -7 & 4 \end{bmatrix} = \begin{bmatrix} 65 & -32 \\ -32 & 17 \end{bmatrix}$$

Finding the eigenvector for AA^T.

$$|A \cdot A' - \lambda I| = 0$$

$$\begin{vmatrix} (65 - \lambda) & -32 \\ -32 & (17 - \lambda) \end{vmatrix} = 0$$

$$\therefore (65 - \lambda) \times (17 - \lambda) - (-32) \times (-32) = 0$$

$$\therefore \left(1105 - 82\lambda + \lambda^2\right) - 1024 = 0$$

$$\therefore \left(\lambda^2 - 82\lambda + 81\right) = 0$$

$$\therefore (\lambda - 1)(\lambda - 81) = 0$$

$$\therefore (\lambda - 1) = 0 \text{ or } (\lambda - 81) = 0$$

∴ The eigenvalues of the matrix A·A′ are given by λ = 1, 81.

Now,

Eigenvectors for λ = 81 are:

$$v_1 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

Eigenvectors for λ = 1 are:

$$v_2 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

For Eigenvector-1 $(-2, 1)$, Length L = $\sqrt{(-2)^2 + 1^2} = 2.236$

So, normalizing gives $u_1 = \left(\frac{-2}{2.236}, \frac{1}{2.236}\right) = (-0.894, 0.447)$

For Eigenvector-2 $(0.5, 1)$, Length L = $\sqrt{0.5^2 + 1^2} = 1.118$

So, normalizing gives $u_2 = \left(\frac{0.5}{1.118}, \frac{1}{1.118}\right) = (0.447, 0.894)$

Similarly, we can find the eigenvectors for A^TA as:

Eigenvectors for λ = 81 are:

$$v_1 = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

Eigenvectors for λ = 1 are:

$$v_2 = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

For Eigenvector-1 (0.5, 1), Length L = $\sqrt{0.5^2 + 1^2} = 1.118$

So, normalizing gives $v_1 = \left(\frac{0.5}{1.118}, \frac{1}{1.118}\right) = (0.447, 0.894)$

For Eigenvector-2 (- 2, 1), Length L = $\sqrt{(-2)^2 + 1^2} = 2.236$

So, normalizing gives $v_2 = \left(\frac{-2}{2.236}, \frac{1}{2.236}\right) = (-0.894, 0.447)$

Using these values, we can write the solution.

$$\therefore \Sigma = \begin{bmatrix} \sqrt{81} & 0 \\ 0 & \sqrt{1} \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\therefore U = \begin{bmatrix} u_1, u_2 \end{bmatrix} = \begin{bmatrix} -0.894 & 0.447 \\ 0.447 & 0.894 \end{bmatrix}$$

V is found using formula $v_i = \frac{1}{\sigma_i} A^T \cdot u_i$

$$\therefore V = \begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$$

Or

$$\therefore \Sigma = \begin{bmatrix} \sqrt{81} & 0 \\ 0 & \sqrt{1} \end{bmatrix} = \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\therefore V = \begin{bmatrix} v_1, v_2 \end{bmatrix} = \begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$$

$$U \text{ is found using formula } u_i = \frac{1}{\sigma_i} A \cdot v_i$$

$$\therefore U = \begin{bmatrix} -0.894 & 0.447 \\ 0.447 & 0.894 \end{bmatrix}$$

So, the final SVD of matrix A = $\begin{bmatrix} -0.894 & 0.447 \\ 0.447 & 0.894 \end{bmatrix} \begin{bmatrix} 9 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.447 & -0.894 \\ 0.894 & 0.447 \end{bmatrix}$.

Singular Value Decomposition Applications:

Some of the applications of singular value decomposition are listed below:

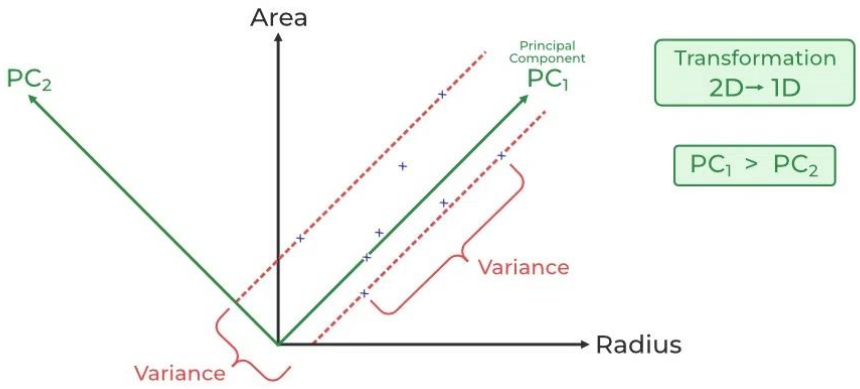
- SVD has some fascinating algebraic characteristics and conveys relevant geometrical and theoretical insights regarding linear transformations.
- SVD has some critical applications in data science too.
- Mathematical applications of the SVD involve calculating the matrix approximation, rank of a matrix and so on.
- The SVD is also greatly useful in science and engineering.
- It has some applications of statistics, for example, least-squares fitting of data and process control.

Principal Component Analysis (PCA):

Principal Component Analysis (PCA) technique was introduced by the mathematician **Karl Pearson** in 1901**.** It works on the condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.

- Principal Component Analysis (PCA) is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover,
- Principal Component Analysis (PCA) is an unsupervised learning algorithm technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.
- The main goal of Principal Component Analysis (PCA) is to reduce the dimensionality of a dataset while preserving the most important patterns or relationships between the variables without any prior knowledge of the target variables.

Principal Component Analysis (PCA) is used to reduce the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables, retaining most of the sample's information, and useful for the regression and classification of data.



Principal Component Analysis

1. Principal Component Analysis (PCA) is a technique for dimensionality reduction that identifies a set of orthogonal axes, called principal components, that capture the maximum variance in the data. The principal components are linear combinations of the original variables in the dataset and are ordered in decreasing order of importance. The total variance captured by all the principal components is equal to the total variance in the original dataset.

2. The first principal component captures the most variation in the data, but the second principal component captures the maximum variance that is orthogonal to the first principal component, and so on.
3. Principal Component Analysis can be used for a variety of purposes, including data visualization, feature selection, and data compression. In data visualization, PCA can be used to plot high-dimensional data in two or three dimensions, making it easier to interpret. In feature selection, PCA can be used to identify the most important variables in a dataset. In data compression, PCA can be used to reduce the size of a dataset without losing important information.
4. In Principal Component Analysis, it is assumed that the information is carried in the variance of the features, that is, the higher the variation in a feature, the more information that features carries.

Principal Components in PCA:

As described above, the transformed new features or the output of PCA are the Principal Components. The number of these PCs are either equal to or less than the original features present in the dataset. Some properties of these principal components are given below:

- The principal component must be the linear combination of the original features.
- These components are orthogonal, i.e., the correlation between a pair of variables is zero.
- The importance of each component decreases when going to 1 to n, it means the 1 PC has the most importance, and n PC will have the least importance.

Step-By-Step Explanation of PCA:

Step 1: Standardization

First, we need to standardize our dataset to ensure that each variable has a mean of 0 and a standard deviation of 1.

$$Z = \frac{X-\mu}{\sigma}$$

Here,

- μ is the mean of independent features $\mu = \{\mu_1, \mu_2, \cdots, \mu_m\}$
- σ is the standard deviation of independent features $\sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_m\}$

Step 2: Covariance Matrix Computation

Covariance measures the strength of joint variability between two or more variables, indicating how much they change in relation to each other. To find the covariance we can use the formula:

$$cov(x1, x2) = \frac{\sum_{i=1}^{n}(x1_i - \bar{x1})(x2_i - \bar{x2})}{n-1}$$

The value of covariance can be positive, negative, or zeros.

- Positive: As the x1 increases x2 also increases.
- Negative: As the x1 increases x2 also decreases.
- Zeros: No direct relation

Step 3: Compute Eigenvalues and Eigenvectors of Covariance Matrix to Identify Principal Components

Let A be a square n*n matrix and X be a non-zero vector for which

$$AX = \lambda X$$

for some scalar values λ. then λ is known as the eigenvalue of matrix A and X is known as the eigenvector of matrix A for the corresponding eigenvalue.

It can also be written as:

$$AX - \lambda X = 0$$
$$(A - \lambda I)X = 0$$

where I is the identity matrix of the same shape as matrix A. And the above conditions will be true only if (A – λI) will be non-invertible (i.e. singular matrix). That means,

$$|A - \lambda I| = 0$$

From the above equation, we can find the eigenvalues λ, and therefore corresponding eigenvector can be found using the equation $AX = \lambda X$.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Step 4: Create a Feature Vector

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance.

In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call ***Feature vector***.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only ***p*** eigenvectors (components) out of ***n***, the final data set will have only ***p*** dimensions.

Step 5: Recast the Data Along the Principal Components Axes

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

Example Dataset:

Consider the following 2-dimensional dataset:

Sample	***x*1**	***x*2**
1	2	3
2	3	3
3	3	4
4	5	5
5	5	6
6	6	7

Solution:

$$X = \begin{bmatrix} 2 & 3 \\ 3 & 3 \\ 3 & 4 \\ 5 & 5 \\ 5 & 6 \\ 6 & 7 \end{bmatrix}$$

Step 1: Standardize the Data

First, calculate the mean of each feature:

$$\text{mean}(x_1) = \frac{2+3+3+5+5+6}{6} = \frac{24}{6} = 4$$

$$\text{mean}(x_2) = \frac{3+3+4+5+6+7}{6} = \frac{28}{6} \approx 4.67$$

Center the data by subtracting the mean of each feature:

$$X_{\text{centered}} = X - \mu = \begin{bmatrix} 2 & 3 \\ 3 & 3 \\ 3 & 4 \\ 5 & 5 \\ 5 & 6 \\ 6 & 7 \end{bmatrix} - \begin{bmatrix} 4 & 4.67 \end{bmatrix} = \begin{bmatrix} -2 & -1.67 \\ -1 & -1.67 \\ -1 & -0.67 \\ 1 & 0.33 \\ 1 & 1.33 \\ 2 & 2.33 \end{bmatrix}$$

Step 2: Calculate the Covariance Matrix

The covariance matrix captures how the two variables vary together. It is computed as:

$$\Sigma = \frac{1}{n-1} X_{\text{centered}}^T X_{\text{centered}}$$

$$\Sigma = \frac{1}{5} \begin{bmatrix} -2 & -1 & -1 & 1 & 1 & 2 \\ -1.67 & -1.67 & -0.67 & 0.33 & 1.33 & 2.33 \end{bmatrix} \begin{bmatrix} -2 & -1.67 \\ -1 & -1.67 \\ -1 & -0.67 \\ 1 & 0.33 \\ 1 & 1.33 \\ 2 & 2.33 \end{bmatrix}$$

$$\Sigma = \frac{1}{5} \begin{bmatrix} 14 & 13 \\ 13 & 11.33 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 2.8 & 2.6 \\ 2.6 & 2.267 \end{bmatrix}$$

Step 3: Compute the Eigenvalues and Eigenvectors

The eigenvalues and eigenvectors of the covariance matrix are calculated by solving the equation:

$$\Sigma v = \lambda v$$

This yields the eigenvalues (λ) and eigenvectors (v):

$$\begin{vmatrix} 2.8 - \lambda & 2.6 \\ 2.6 & 2.267 - \lambda \end{vmatrix} = 0$$

Solving this characteristic equation gives us the eigenvalues:

$$\lambda_1 \approx 4.937, \quad \lambda_2 \approx 0.130$$

The corresponding eigenvectors are:

$$v_1 = \begin{bmatrix} -0.687 \\ -0.726 \end{bmatrix}, \quad v_2 = \begin{bmatrix} -0.726 \\ 0.687 \end{bmatrix}$$

Step 4: Sort Eigenvalues and Eigenvectors

Sort the eigenvalues in descending order and arrange the corresponding eigenvectors accordingly. Here, $\lambda_1 > \lambda_2$, so we have:

$$\text{Eigenvalue} = 4.937 \quad \text{with eigenvector} \quad v_1 = \begin{bmatrix} -0.687 \\ -0.726 \end{bmatrix}$$

$$\text{Eigenvalue} = 0.130 \quad \text{with eigenvector} \quad v_2 = \begin{bmatrix} -0.726 \\ 0.687 \end{bmatrix}$$

Step 5: Select Principal Components

We choose the top k eigenvectors. If $k = 1$, we select v_1:

$$\text{Principal Component} = \begin{bmatrix} -0.687 \\ -0.726 \end{bmatrix}$$

Step 6: Transform the Data

Finally, project the centered data onto the new feature space (principal component):

$$X_{\text{PCA}} = X_{\text{centered}} \cdot v_1$$

$$X_{\text{PCA}} = \begin{bmatrix} -2 & -1.67 \\ -1 & -1.67 \\ -1 & -0.67 \\ 1 & 0.33 \\ 1 & 1.33 \\ 2 & 2.33 \end{bmatrix} \begin{bmatrix} -0.687 \\ -0.726 \end{bmatrix} = \begin{bmatrix} 2.585 \\ 1.898 \\ 1.172 \\ -0.929 \\ -1.656 \\ -3.069 \end{bmatrix}$$