SJSU SAN JOSÉ STATE UNIVERSITY

DATA 220: Mathematical Methods for Data Analytics

Dr. Mohammad Masum

Linear Algebra in Data Science

- Linear algebra is a branch of mathematics that deals with linear equations and their representations through matrices and vectors
- Linear algebra plays a crucial role in DS & ML algorithms involve working with vectors and matrices
- Understanding linear algebra allows us to efficiently perform computations on large datasets, manipulate images and audio signals, and solve optimization problems.
- Some common applications of linear algebra in data science include:
  - Principal Component Analysis (PCA) for dimensionality reduction
  - Singular Value Decomposition (SVD) for matrix factorization
  - Linear regression for fitting models to data
  - Image processing and computer vision
  - Natural language processing and text analysis
  - Network analysis and graph theory

Vectors

- A vector is ordered list of numbers
  - numbers in the list are elements (entries/coefficients)
- Vectors can be manipulated using Linear Algebra operations such as addition, multiplications and transformations
- Vector size – number of elements
  - Vector of size n is n-vector
- Numbers are called scalers

$$\begin{bmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{bmatrix} \text{ or } \begin{pmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{pmatrix}$$

or $(-1.1, 0, 3.6, -7.2)$

https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/vectors/v/vector-introduction-linear-algebra

Vectors

- Vectors notation
  - In general, by a small letter *a*
  - ith element *of* n -vector a is denoted by a_i
    - *i is the index*
    - *Indices run from $i = 1$ to $i = n$*
  - $a_1 = -1.1; a_4 = -7.2$
  - *We can call two equal vectors a and b are equal if*
    - *For all $i, a_i = b_i$ or we can write generally, $a = b$*

$$\begin{bmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{bmatrix} \text{ or } \begin{pmatrix} -1.1 \\ 0.0 \\ 3.6 \\ -7.2 \end{pmatrix}$$

or $(-1.1, 0, 3.6, -7.2)$

Block Vectors

- We can concatenate vectors
  - a is the block vector with block entries b, c, d with sizes m,n,p
  - Size of a is m + n + p

$$a = \begin{bmatrix} b \\ c \\ d \end{bmatrix}$$

$$a = (b_1, b_2, \ldots, b_m, c_1, c_2, \ldots, c_n, d_1, d_2, \ldots, d_p)$$

Zeros, ones and unit Vectors

- Zero vector: $n - vector\ with\ all\ zero\ entries$
  - *Denoted as* ***0***
- *One vector:* $n - vector\ with\ all\ entries$ ***1*** ***or*** $\mathbf{1}_{\boldsymbol{n}}$
- *Unit vector: has one entry 1 and all other zeros*
  - *Denoted as* $e_i\ where\ ith\ entry\ is\ 1$

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Sparse Vectors

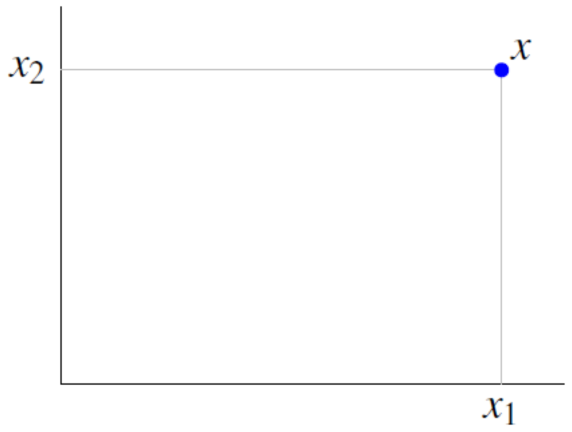
- A vector is sparse if many of its entries are zero
- Can be stored and manipulated efficiently on a computer
  - can greatly reduce the computational cost and memory requirements of processing high-dimensional data. By representing data as sparse vectors, we can efficiently perform operations such as dot products, matrix multiplications, and distance calculations
- Examples of sparse vectors can include:
  - Text data, where each word or token can be represented as a feature in a high-dimensional vector space
  - Images, where each pixel or region can be represented as a feature in a high-dimensional vector space

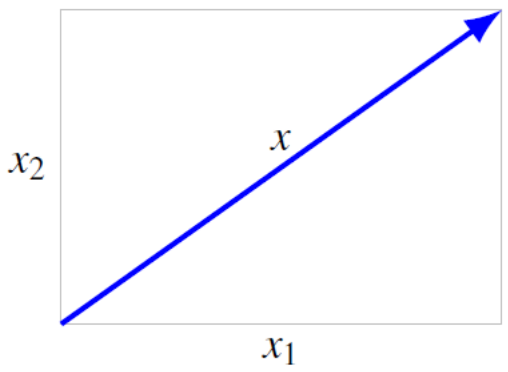
Features / Words	and	ball	Cat	Dog	go	hates	it	loves	out	play	to	with
Cat	0	0	**1**	0	0	0	0	0	0	0	0	0
Loves	0	0	0	0	0	0	0	**1**	0	0	0	0
To	0	0	0	0	0	0	0	0	0	0	**1**	0
Play	0	0	0	0	0	0	0	0	0	**1**	0	0
With	0	0	0	0	0	0	0	0	0	0	0	**1**
A	0	0	0	0	0	0	0	0	0	0	0	0
Ball	0	**1**	0	0	0	0	0	0	0	0	0	0

Image source - https://www.einfochips.com/blog/nlp-text-vectorization/

Location

- 2-vector $(x_1,\ x_2)$ can represent a location or a displacement in 2-D





Word Count Vector

- Word Count vector
  - represent text data as a numerical vector
  - It is a type of bag-of-words (BoW) representation, which simply counts the frequency of words in a document without considering their order
- Suppose you have the following two sentences: "The cat in the hat.“ & "The dog chased the cat."
  - listing all the unique words in the corpus, which in this case is ["the", "cat", "in", "hat", "dog", "chased"].
  - count the frequency of each word in the vocabulary and create a vector with those counts:
    - [2, 1, 1, 1, 0, 0] for "The cat in the hat."
    - [1, 1, 0, 0, 1, 1] for "The dog chased the cat.“
- For example, in text classification, you can use word count vectors to represent documents and train a classifier to predict the category of a new document

Vector addition

- Adding one vector to another vector
  - Pointwise addition
    - $a = [a_1, a_2, \ldots, a_n]$ and $b = [b_1, b_2, \ldots, b_3]$
    - $a + b = [a_1 + b_1, a_2 + b_2, \ldots . a_n + b_n]$
- Vector addition is commutative: $a + b = b + a$
- Vector addition is associative: $a + (b + c) = (a + b) + c$

$$\begin{bmatrix} 0 \\ 7 \\ 3 \end{bmatrix} + \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 9 \\ 3 \end{bmatrix}$$

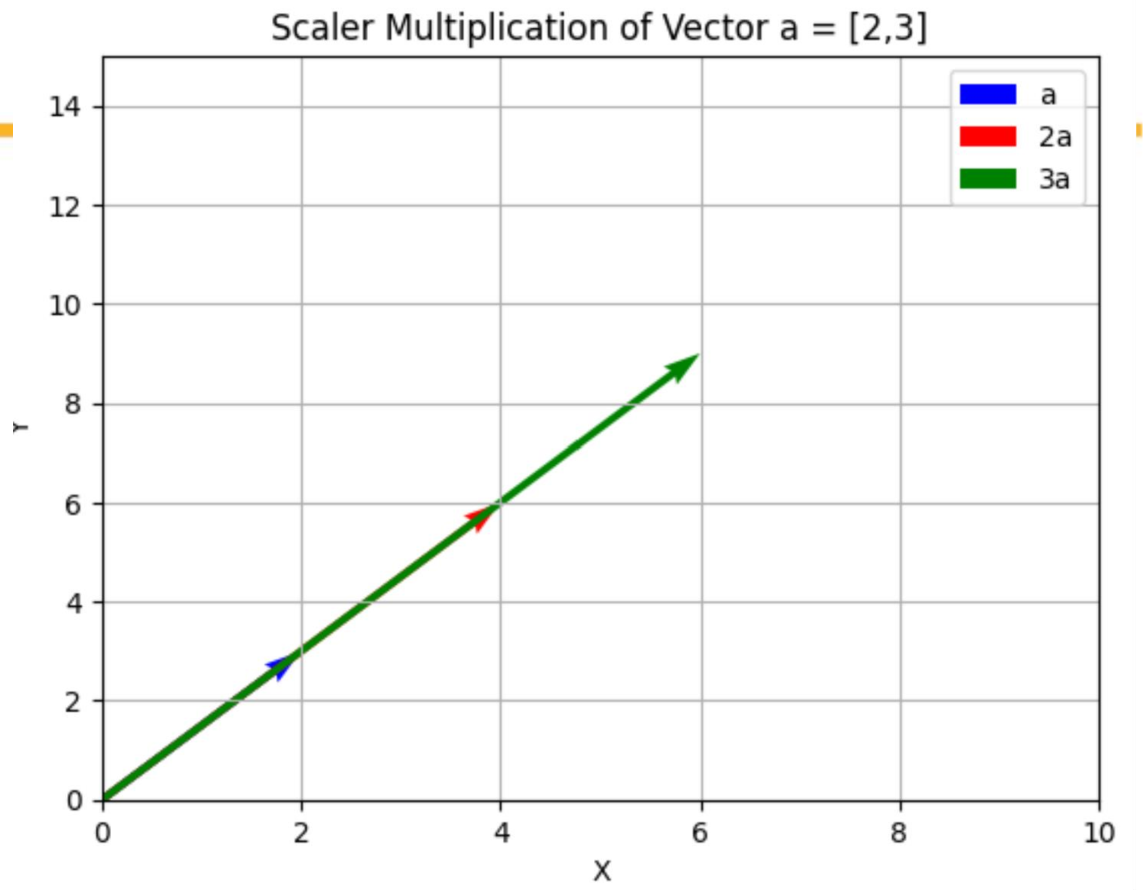
Vector Subtraction

- Subtracting one vector from another
  - Pointwise subtraction
  - $a = [a_1, a_2, \ldots, a_n]$ and $b = [b_1, b_2, \ldots, b_3]$
  - $a - b = [a_1 - b_1, a_2 - b_2, \ldots . a_n - b_n]$
- The order of subtraction matters: $a - b \neq b - a$
- Subtracting a vector is equivalent to adding the negation of that vector.

SJSU SAN JOSÉ STATE UNIVERSITY

Scaler Vector multiplication

- Scalar-vector multiplication is the operation of multiplying a scalar quantity by a vector
  - $\beta a = [\beta a_1, \beta a_2, \dots, \beta a_n]$; where $\beta\ is\ scaler\ and\ a\ is\ a\ vector$
- The resulting vector has the **same direction** as the original vector but its magnitude is multiplied by the scalar value



$$(-2)\begin{bmatrix} 1 \\ 9 \\ 6 \end{bmatrix} = \begin{bmatrix} -2 \\ -18 \\ -12 \end{bmatrix}$$

Scaler Vector multiplication

- Distributive Property: If a scalar k multiplies a vector (u + v), the result is equal to (k * u) + (k * v)
  - the scalar can be distributed across the sum of two vectors.
- Associative Property: If k and m are scalars and u is a vector, then (k * m) * u = k * (m * u)
  - the order of scalar multiplication does not matter.
- Commutative Property: If k is a scalar and u is a vector, then k * u = u * k.
  - the order of scalar multiplication and vector does not matter.
- Identity Property: If 1 is the scalar for multiplication and u is a vector, then 1 * u = u.
  - scalar multiplication by 1 does not change the vector.
- Zero Property: If 0 is the scalar for multiplication and u is a vector, then 0 * u = 0
  - scalar multiplication by 0 results in a zero vector

Linear Combination

- Linear Combination
  - combining a set of vectors (which can be of any dimension) with scalar coefficients, such that each vector is weighted by a corresponding scalar and then summed together
  - Assume, $a_1, a_2, \ldots . a_n$ are vectors and $\beta_1, \beta_2, \ldots, \beta_n$ are scalers:

$\beta_1 a_1 + \beta_2 a_2 + \ \ldots . \beta_n a_n$ is a linear combination

https://www.3blue1brown.com/lessons/span

Span of a Set of Vectors

- Span of a set of vectors – is the set of all possible linear combinations of those vectors
  - In other words, it is the set of all vectors that can be expressed as a linear combination of the given vectors.
  - For example, suppose we have two vectors in R^2, $v_1 = (1, 0)\ and\ v_2 = (0, 1)$
    - span of these two vectors is the set of all linear combinations of v1 and v2, which can be written as $av_1 + bv_2$, where a and b are scalars
    - resulting set is the entire xy-plane, since any point in the xy-plane can be written as a linear combination of v_1 and v_2

https://www.3blue1brown.com/lessons/span

SJSU SAN JOSÉ STATE UNIVERSITY

Vector – Inner product

$$a^T b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

$$\begin{bmatrix} -1 \\ 2 \\ 2 \end{bmatrix}^T \begin{bmatrix} 1 \\ 0 \\ -3 \end{bmatrix} = (-1)(1) + (2)(0) + (2)(-3) = -7$$

- Inner Product
  - is a way to calculate the angle between two vectors
  - Inner product or dot product of $n - vectors\ a\ and\ b\ is\ a\,.\,b\ \ or\ ,< a, b >$
  - $a.\,b = |a|\ |b| \cos\theta$; where $|a|\ and\ |b|$ are the magnitude (norm) of the vectors $a\ and\ b$

Properties

1. $a^T b = b^T a$
2. $(\gamma a)^T b = \gamma\ (a^T b)$
3. $(a + b)^T\ c\ = a^T c + b^T c$

We can combine above.

$$(a + b)^T (c + d) = a^T c + a^T d\ + b^T c + b^T d$$

Vector – Inner product

$$a^T b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

- picking out ith entry: $e_i^T a = a_i$
- sum of elements of a vector: $\mathbf{1}^T a = a_1 + \ \ldots + a_n$
- sum of squares of a vector entries $a^T a = a_1^2 + \ \ldots . + a_n^2$

Vector Norm

- Vector Norm- the length of a vector
  - Vector norm is non-negative number
  - Distance of the vector from the origin of the vector space
  - Vector norm can be calculated using $L^1 \ \ and \ L^2$ norm

SJSU SAN JOSÉ STATE UNIVERSITY

Vector Norm

- Vector L^1 norm
  - Also known as Manhattan Norm
  - Sum of the absolute vector values
  - Applications in ML
    - Mainly used for regularization

$$L^1\,(v) = \|v\|_1 = |x_1| + |x_2| + |x_3| + \cdots + |x_n|$$

```
import numpy as np
```

```
v1 = np.array([1,2,3,4,5])
norm1_v1 = np.linalg.norm(v1,1)
print("L1 norm of vector v1: {}".format(norm1_v1))
```

```
L1 norm of vector v1: 15.0
```

Vector Norm

- Vector L^1 norm
  - Also known as Manhattan Norm
  - Sum of the absolute vector values

$$L^1\,(v) = \|v\|_1 = |x_1| + |x_2| + |x_3| + \cdots + |x_n|$$

```
import numpy as np
```

```
v1 = np.array([1,2,3,4,5])
norm1_v1 = np.linalg.norm(v1,1)
print("L1 norm of vector v1: {}".format(norm1_v1))
```

```
L1 norm of vector v1: 15.0
```

```
v2 = np.array([1,-2,-3,4,-5])
norm1_v2 = np.linalg.norm(v2,1)
print("L1 norm of vector v2: {}".format(norm1_v2))
```

```
L1 norm of vector v2: 15.0
```

Vector Norm

- Vector L^2 norm
  - Distance of the vector coordinate from the origin of the vector space
  - Also known as, Euclidean Norm
  - Always non-negative number
  - Applications in ML
    - Used for Regularization

$$L^2(v) = \|v\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots\ldots + x_n^2}$$

Vector Norm

$$L^2(v) = \|v\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots\ldots + x_n^2}$$

```
[12]
v1 = np.array([1,2,3,4,5])
norm2_v1 = np.linalg.norm(v1,2)
print("L2 norm of vector v1: {}".format(norm2_v1))
```

```
L2 norm of vector v1: 7.416198487095663
```

```
[13]
v2 = np.array([1,2,-3,-4,5])
norm2_v2 = np.linalg.norm(v1,2)
print("L2 norm of vector v2: {}".format(norm2_v1))
```

```
L2 norm of vector v2: 7.416198487095663
```

SJSU SAN JOSÉ STATE UNIVERSITY

$$L^{inf}(v) = \|v\|_{inf} = \max\ |x_1|, |x_2|, \ldots, |x_n|$$

Vector Norm

- Vector Max norm
  - Maximum absolute value of the vector
  - Applications in ML
    - NN weights

```
15]   1 v1 = np.array([1,2,3,4,5])
      2 norm_v1 = np.linalg.norm(v1, np.inf)
      3 print("inf norm of vector v1: {}".format(norm_v1))
```

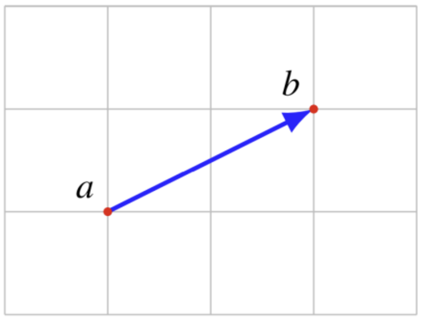
```
inf norm of vector v1: 5.0
```

```
[17]  1 v2 = np.array([1,2,3,-4, -5])
      2 norm_v2 = np.linalg.norm(v2, np.inf)
      3 print("inf norm of vector v2: {}".format(norm_v2))
```

```
inf norm of vector v2: 5.0
```

Vector distance

$$dist(a, b)$$



- Vector Distance
  - distance between two points (samples/observations/rows/data) in a multi-dimensional space
  - determine the similarity between vectors
    - Essential tasks in DS & ML – clustering and classification
  - Several ways to calculate the distance
    - Euclidean distance
    - Manhattan distance
    - Cosine similarity (primarily, used in Text Analysis)

Vector distance

- Euclidean distance
  - $dist(a, b) = \sqrt{(a_1 - b_1)^2 \;+\; \ldots + (a_n - b_n)^2}$
- Manhattan distance
  - Sum of the absolute differences between the coordinates of two points
  - $dist(a, b) = |a_1 \; - b_1| + \; \ldots + |a_n - b_n|$

Vector distance

Cosine similarity

- Cosine similarity is a measure of the similarity between two non-zero vectors of an inner product space
- It is the cosine of the angle between the two vectors in the multidimensional space.

Suppose a and b two non-zero vectors, then

$$a.b = \|a\| \, \|b\| \cos\theta$$

Cosine similarity = $S_c\,(a, b) = \cos\theta = \dfrac{a.b}{\|a\| \, \|b\|} = \dfrac{a^T b}{\|a\| \, \|b\|}$

Similarity scores: [-1,1]
-1: exactly opposite
1: exactly the same
0: decorrelation
while in-between values indicate intermediate similarity or dissimilarity.

Cosine distance = 1- cosine similarity

Vector distance

	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

```python
import numpy as np
from numpy.linalg import norm
```

[8]

```python
doc_1 = np.array([1,1,1,0,0,1,0])
doc_2 = np.array([1,1,0,0,0,1,0])
doc_3 = np.array([1,1,0,1,0,1,0])
doc_4 = np.array([0,2,0,0,1,2,1])
doc_5 = np.array([1,1,0,0,0,1,0])
```

[12]

```python
# compute cosine similarity
S = np.dot(doc_1,doc_2)/(norm(doc_1)*norm(doc_2))
print("Cosine Similarity: {}".format(S))
```

```
Cosine Similarity: 0.8660254037844387
```

Linear Dependence & Independence

- A set of vectors are **linearly independent** if none of the vectors can be written as a linear combination of the others
  - In other words, a set of vectors is linearly independent if no vector in the set is redundant and can be expressed in terms of the other vectors in the set
  - Two vectors are linearly independent if they point in different directions in space
- A set of vectors is **linearly dependent** if at least one of the vectors in the set can be expressed as a linear combination of the other vectors in the set

Linear Dependence & Independence

- Mathematically
  - A set of vectors $\{v_1, v_2, \dots, v_n\}$ is linearly independent if the only solution to the equation
    - $a_1 v_1 + \dots + a_n v_n = 0$ is $a_1 = a_2 = \dots = a_n = 0$, where $a_1, a_2, \dots, a_n$ are scalar coefficients
  - A set of vectors $\{v_1, v_2, \dots, v_n\}$ is linearly dependent if there exist scalar coefficients $a_1, a_2, \dots, a_n$; **not all zero**, such that $a_1 v_1 + \dots + a_n v_n = 0$

- Linear independence –
  - feature selection, where linearly dependent features can be removed to simplify the analysis and improve the performance of the model

SJSU SAN JOSÉ STATE UNIVERSITY

Linear Dependence & Independence

Example 1

$$v1 = [1\ 0\ 0]$$
$$v2 = [0\ 1\ 0]$$
$$v3 = [0\ 0\ 1]$$

Example 2

$$v1 = [1\ 0\ 0]$$
$$v2 = [2\ 0\ 0]$$
$$v3 = [3\ 0\ 0]$$

Example 3

$$v1 = [1\ 2]$$
$$v2 = [2\ 4]$$
$$v3 = [3\ 6]$$

SJSU SAN JOSÉ STATE UNIVERSITY

Dummy Variable

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

- Dummy variables are indicator variables used to represent categories
- The Dummy Variable Trap arises when dummy variables are **linearly dependent**, leading to multicollinearity issues

```python
# Create dummy variables for the "Embarked" column
embarked_dummies = pd.get_dummies(titanic_data['Embarked'],
                                  prefix='Embarked')
embarked_dummies
```

	Embarked_C	Embarked_Q	Embarked_S
0	0	0	1
1	1	0	0
2	0	0	1
3	0	0	1
4	0	0	1
...	...	...	...

```python
# Create dummy variables for the "Embarked" column
embarked_dummies = pd.get_dummies(titanic_data['Embarked'],
                                  prefix='Embarked',
                                  drop_first= True)
embarked_dummies
```

	Embarked_Q	Embarked_S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1
...	...	...

https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html

Basis

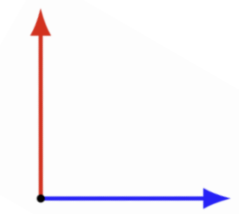
- Basis
  - a set of linearly independent vectors that span a space
  - the number of vectors in a basis is called the dimension of the space
- Mathematically, let, A set of vectors $\{v_1, v_2, \ldots, v_n\}$ is a basis of a vector space V:
  - The set is linearly independent, i.e., no vector in the set can be expressed as a linear combination of the other vectors in the set
  - The set spans V, i.e., for any vector $v \in V$, there exist scalars $a_1, a_2, \ldots, a_n$ such that

$$v = a_1 v_1 + a_2 v_2 + \ldots + a_n v_n$$

Orthonormal Vectors

- Orthonormal vectors are a set of vectors in which each vector is unit-length and orthogonal (perpendicular) to each other
  - a set of orthonormal vectors forms a basis for a vector space
- let's consider a vector space V and A set of n vectors $\{v_1, v_2, \ldots, v_n\}$ in V is said to be orthonormal if the following conditions hold:
  - The magnitude (length) of each vector is 1: $||v_1|| = ||v_2|| = \ldots = ||v_n|| = 1$.
  - Each pair of vectors is orthogonal (perpendicular): $v_i \cdot v_j = 0$ for all $i \neq j$, where $\cdot$ denotes the dot product.

$$\begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}, \quad \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$



Matrix

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}$$

- Matrix
  - Two-dimensional array of number
  - Two matrices are equal if they are same size and corresponding entries are equal
  - $m \times n$ matrix A is
    - Tall if $m > n$
    - Wide if $m < n$
    - Square if $m = n$

```
[26]  1 A = np.array([[1,2,3], [4,5,6], [7,8,9]])
      2 print(A)

[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

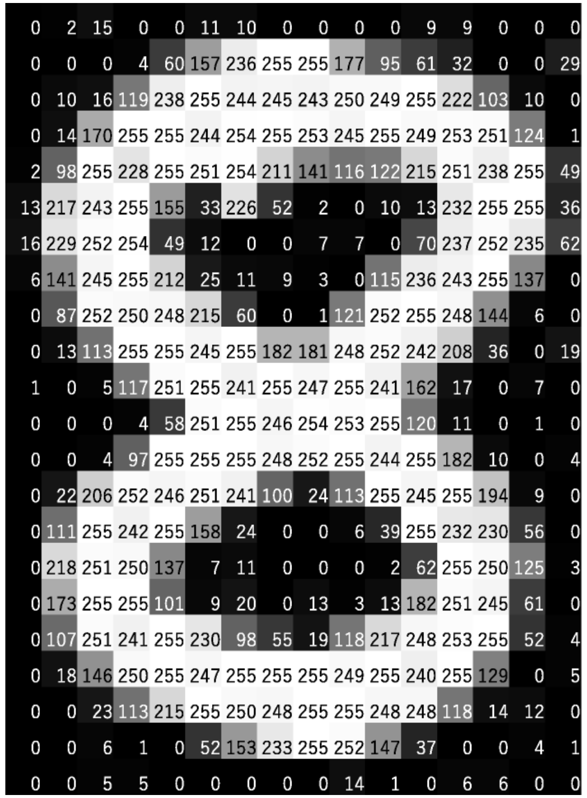
An $n \times 1$ matrix is n-vector

A 1×1 matrix is scaler

An $1 \times n$ matrix is a row vector which is not same as the $n \times 1$ column vector

Example

- Image: X_{ij} pixel values



0	2	15	0	0	11	10	0	0	0	0	9	9	0	0	0
0	0	0	4	60	157	236	255	255	177	95	61	32	0	0	29
0	10	16	119	238	255	244	245	243	250	249	255	222	103	10	0
0	14	170	255	255	244	254	255	253	245	255	249	253	251	124	1
2	98	255	228	255	251	254	211	141	116	122	215	251	238	255	49
13	217	243	255	155	33	226	52	2	0	10	13	232	255	255	36
16	229	252	254	49	12	0	0	7	7	0	70	237	252	235	62
6	141	245	255	212	25	11	9	3	0	115	236	243	255	137	0
0	87	252	250	248	215	60	0	1	121	252	255	248	144	6	0
0	13	113	255	255	245	255	182	181	248	252	242	208	36	0	19
1	0	5	117	251	255	241	255	247	255	241	162	17	0	7	0
0	0	0	4	58	251	255	246	254	253	255	120	11	0	1	0
0	0	4	97	255	255	255	248	252	255	244	255	182	10	0	4
0	22	206	252	246	251	241	100	24	113	255	245	255	194	9	0
0	111	255	242	255	158	24	0	0	6	39	255	232	230	56	0
0	218	251	250	137	7	11	0	0	0	2	62	255	250	125	3
0	173	255	255	101	9	20	0	13	3	13	182	251	245	61	0
0	107	251	241	255	230	98	55	19	118	217	248	253	255	52	4
0	18	146	250	255	247	255	255	255	249	255	240	255	129	0	5
0	0	23	113	215	255	250	248	255	255	248	248	118	14	12	0
0	0	6	1	0	52	153	233	255	252	147	37	0	0	4	1
0	0	5	5	0	0	0	0	0	14	1	0	6	6	0	0

Yamashita, R., Nishio, M., Do, R.K.G. *et al.* Convolutional neural networks: an overview and application in radiology. *Insights Imaging* **9**, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9

Example

- Text data: term-frequency document matrix

	it	is	puppy	cat	pen	a	this
it is a puppy	1	1	1	0	0	1	0
it is a kitten	1	1	0	0	0	1	0
it is a cat	1	1	0	1	0	1	0
that is a dog and this is a pen	0	2	0	0	1	2	1
it is a matrix	1	1	0	0	0	1	0

https://www.oreilly.com/library/view/feature-engineering-for/9781491953235/ch04.html

Matrix

- Matrix Addition
  - Point-wise scaler addition

$$C = \begin{pmatrix} a_{1,1} + b_{1,1} & a_{1,2} + b_{1,2} \\ a_{2,1} + b_{2,1} & a_{2,2} + b_{2,2} \\ a_{3,1} + b_{3,1} & a_{3,2} + b_{3,2} \end{pmatrix}$$

```
[29]  A = np.array([[1,2,3], [4,5,6], [7,8,9]])
      B = np.array([[-11,-2,-3], [-4,5,6], [-7,8,9]])
```

```
[30]  A
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
[31]  B
```

```
array([[-11,  -2,  -3],
       [ -4,   5,   6],
       [ -7,   8,   9]])
```

```
[33]  A + B
```

```
array([[-10,   0,   0],
       [  0,  10,  12],
       [  0,  16,  18]])
```

Matrix

- Matrix Subtraction
  - Point-wise scaler subtraction

$$C = \begin{pmatrix} a_{1,1} - b_{1,1} & a_{1,2} - b_{1,2} \\ a_{2,1} - b_{2,1} & a_{2,2} - b_{2,2} \\ a_{3,1} - b_{3,1} & a_{3,2} - b_{3,2} \end{pmatrix}$$

```
[29]  1 A = np.array([[1,2,3], [4,5,6], [7,8,9]])
      2 B = np.array([[-11,-2,-3], [-4,5,6], [-7,8,9]])
```

```
[30]  1 A
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
[31]  1 B
```

```
array([[-11,  -2,  -3],
       [ -4,   5,   6],
       [ -7,   8,   9]])
```

```
1 A - B
```

```
array([[12,  4,  6],
       [ 8,  0,  0],
       [14,  0,  0]])
```

Matrix

- Matrix Multiplication (Hadamard Product)
  - Element wise matrix multiplication
  - Two matrices shape should be identical

$$C = \begin{pmatrix} a_{1,1} \times b_{1,1} & a_{1,2} \times b_{1,2} \\ a_{2,1} \times b_{2,1} & a_{2,2} \times b_{2,2} \\ a_{3,1} \times b_{3,1} & a_{3,2} \times b_{3,2} \end{pmatrix}$$

```
[30]  1 A

array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
[31]  1 B

array([[-11,  -2,  -3],
       [ -4,   5,   6],
       [ -7,   8,   9]])
```

```
[36]  1 C = A * B
      2 print(C)

[[-11  -4  -9]
 [-16  25  36]
 [-49  64  81]]
```

Matrix

- Matrix Division
  - Element wise division
  - Two matrices shape should be identical

$$C = \begin{pmatrix} \frac{a_{1,1}}{b_{1,1}} & \frac{a_{1,2}}{b_{1,2}} \\ \frac{a_{2,1}}{b_{2,1}} & \frac{a_{2,2}}{b_{2,2}} \\ \frac{a_{3,1}}{b_{3,1}} & \frac{a_{3,2}}{b_{3,2}} \end{pmatrix}$$

```
[30]  1 A
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
[31]  1 B
```

```
array([[-11,  -2,  -3],
       [ -4,   5,   6],
       [ -7,   8,   9]])
```

```
[37]  1 C = A / B
      2 print(C)
```

```
[[-0.09090909 -1.         -1.        ]
 [-1.          1.          1.        ]
 [-1.          1.          1.        ]]
```

Matrix

$$C(m,k) = A(m,n) \cdot B(n,k)$$

- Dot Product
  - The number of columns in matrix A should be equal to number of rows in Matrix B

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}$$

$$C = \begin{pmatrix} a_{1,1} \times b_{1,1} + a_{1,2} \times b_{2,1}, a_{1,1} \times b_{1,2} + a_{1,2} \times b_{2,2} \\ a_{2,1} \times b_{1,1} + a_{2,2} \times b_{2,1}, a_{2,1} \times b_{1,2} + a_{2,2} \times b_{2,2} \\ a_{3,1} \times b_{1,1} + a_{3,2} \times b_{2,1}, a_{3,1} \times b_{1,2} + a_{3,2} \times b_{2,2} \end{pmatrix}$$

$$B = \begin{pmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{pmatrix}$$

Matrix

- Dot Product
  - The number of columns in matrix A should be equal to number of rows in Matrix B

```
[46]  1 A

array([[1, 2],
       [4, 5],
       [7, 8]])

[47]  1 B

array([[-11,  -2,  -3],
       [ -7,   8,   9]])

[49]  1 C = A@B
      2 print(C)

[[ -25   14   15]
 [ -79   32   33]
 [-133   50   51]]
```

Matrix

- Matrix vector Multiplication
  - Must follow the rule of matrix multiplication
  - Number of columns in matrix must match with the number of elements in the vector

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \\ a_{3,1} & a_{3,2} \end{pmatrix}$$

$$v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

$$c = \begin{pmatrix} a_{1,1} \times v_1 + a_{1,2} \times v_2 \\ a_{2,1} \times v_1 + a_{2,2} \times v_2 \\ a_{3,1} \times v_1 + a_{3,2} \times v_2 \end{pmatrix}$$

[55]
```
A
```

```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

[56]
```
v
```

```
array([-7,  8])
```

[57]
```
C = A@v
print(C)
```

```
[ 9 12 15]
```

Matrix

- Matrix scalar Multiplication

$$C = \begin{pmatrix} a_{1,1} \times b & a_{1,2} \times b \\ a_{2,1} \times b & a_{2,2} \times b \\ a_{3,1} \times b & a_{3,2} \times b \end{pmatrix}$$

```
[55]  1 A
```

```
array([[1, 2],
       [4, 5],
       [7, 8]])
```

```
[58]  1 b = 10
```

```
[60]  1 C= A*b
      2 print(C)
```

```
[[10 20]
 [40 50]
 [70 80]]
```

Matrix

- Inverse Matrix
  - A $n \times n$ square matrix A is invertible (non-singular) if there exist an $n \times n$ square matrix: $AB = BA = I_n$

- Properties for Invertible matrix
  - $(A^{-1})^{-1} = A$
  - $(kA)^{-1} = k^{-1}A^{-1}$ $for\ nonzero\ k$
  - $(A^T)^{-1} = \left(A^{-1}\right)^T$
  - $\det A^{-1} = (\det A)^{-1}$
  - For any invertible $n \times n$ matrices A and B:
    - $(AB)^{-1} = (BA)^{-1}$

SJSU SAN JOSÉ STATE UNIVERSITY

Matrix

There are many ways to find inverse of a matrix:

- The adjugate of a matrix A can be used to find the inverse of A:

$$A^{-1} = \frac{1}{\det A}\ (adj\ (A))$$

- Gaussian Elimination

Matrix

- Gaussian Elimination
  - Also known as row reduction
  - Algorithm for solving systems of linear equations
  - Consists of sequence of operation performed on the corresponding matrix of coefficients
    - Apply the sequence of elementary row operations to convert the matrix into a **row reduced echelon form**
    - Three types of elementary row operations
      1. Swapping two rows
      2. Multiplying a row by non-zero number
      3. Adding a multiple of one row to another row
  - Can also be used for compute the rank of a matrix, determinant of a square matrix and inverse of an invertible matrix

Matrix

- Find inverse of an invertible matrix A by Gaussian Elimination process

```
A = np.array([[1,2,-1], [-2,0,1], [1,-1,0]])
A
```

```
array([[ 1,  2, -1],
       [-2,  0,  1],
       [ 1, -1,  0]])
```

Matrix

- Find inverse of an invertible matrix A by Gaussian Elimination process

```
[45]  1 A = np.array([[1,2,-1], [-2,0,1], [1,-1,0]])
      2 A
```

```
array([[ 1,  2, -1],
       [-2,  0,  1],
       [ 1, -1,  0]])
```

```
[49]  1 B = np.linalg.inv(A)
      2 B
```

```
array([[1., 1., 2.],
       [1., 1., 1.],
       [2., 3., 4.]])
```

```
[50]  1 A@B
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

Matrix

- A square matrix, that is not invertible is singular matrix
  - A square matrix is singular if and only if its **determinant is zero**

Matrix

- Systems of linear equations
  - Set of m linear equations in n variables $x_1, \ldots, x_n$

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n &= b_2 \\ &\vdots \\ A_{m1}x_1 + A_{m2}x_2 + \cdots + A_{mn}x_n &= b_m \end{aligned}$$

$$\boldsymbol{Ax} = \boldsymbol{b}$$

Classification of system of linear equations:

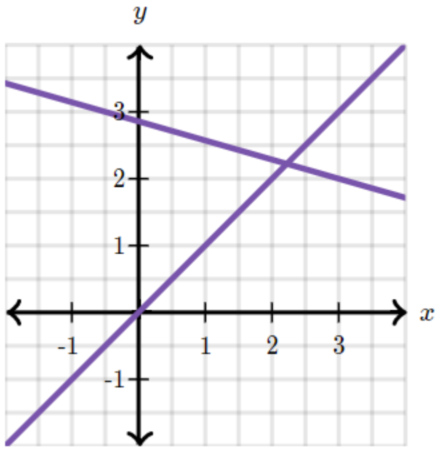
1. under-determined $if\ m < n$
2. Square $if\ m = n$
3. Over-determined $if\ m > n$

x is called a solution if $Ax = b$
Depending on the matrix A and vector b, the system have no solution, one solution and many solutions

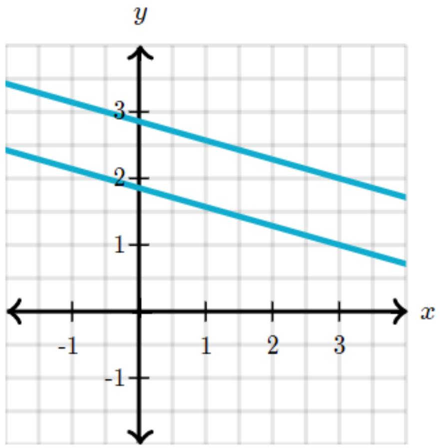
We can use the RREF to determine whether the system has no, one or many solutions

Matrix

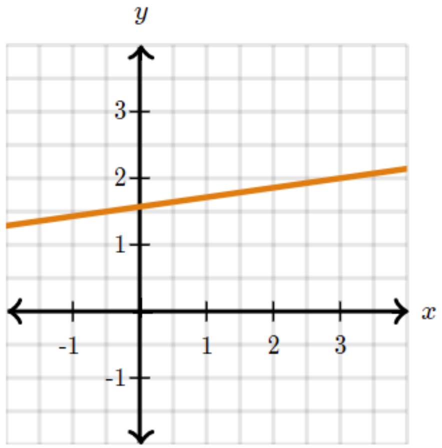
- Systems of linear equations
  - Set of m linear equations in n variables $x_1, \dots, x_n$



Unique solution



No solution



Infinite solution

https://www.khanacademy.org/math/algebra/x2f8bb11595b61c86:systems-of-equations/x2f8bb11595b61c86:number-of-solutions-to-systems-of-equations/a/number-of-solutions-to-system-of-equations-review

Matrix

System of linear equations	$\begin{aligned} 2x + 8y + 4z &= 2 \\ 2x + 5y + z &= 5 \\ 4x + 10y - z &= 1. \end{aligned}$	$\begin{gathered} x + y + z = 2 \\ y - 3z = 1 \\ 2x + y + 5z = 0 \end{gathered}$	$\begin{gathered} -3x - 5y + 36z = 10 \\ -x + 7z = 5 \\ x + y - 10z = -4 \end{gathered}$
Augmented matrix	$\begin{bmatrix} 2 & 8 & 4 & 2 \\ 2 & 5 & 1 & 5 \\ 4 & 10 & -1 & 1 \end{bmatrix}$	$\left\lvert\begin{array}{ccc\|c} 1 & 1 & 1 & 2 \\ 0 & 1 & -3 & 1 \\ 2 & 1 & 5 & 0 \end{array}\right\rvert$	$\left\lvert\begin{array}{ccc\|c} -3 & -5 & 36 & 10 \\ -1 & 0 & 7 & 5 \\ 1 & 1 & -10 & -4 \end{array}\right\rvert$
RREF	$\begin{bmatrix} 1 & 0 & 0 & 11 \\ 0 & 1 & 0 & -4 \\ 0 & 0 & 1 & 3 \end{bmatrix}$	$\left\lvert\begin{array}{ccc\|c} 1 & 0 & 4 & 1 \\ 0 & 1 & -3 & 1 \\ 0 & 0 & 0 & -3 \end{array}\right\rvert$	$\left\lvert\begin{array}{ccc\|c} 1 & 0 & -7 & -5 \\ 0 & 2 & -3 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right\rvert$
	Unique solution	No solution	Infinite solution

Matrix

- unique solution

```
[32]

# importing library sympy
from sympy import symbols, Eq, solve


x, y, z = symbols('x,y,z')

# defining equations
eq1 = Eq((2*x + 8* y+ 4 * z), 2)
eq2 = Eq((2*x + 5*y + z), 5)
eq3 = Eq((4*x+ 10*y - z), 1)


print(solve((eq1, eq2, eq3), (x, y, z)))
```

```
{x: 11, y: -4, z: 3}
```

SJSU SAN JOSÉ STATE UNIVERSITY

Matrix

- No solution

```
[25]
from scipy.linalg import solve

A = [[1, 1, 1], [0, 1, -3], [2, 1, 5]]
b = [[2], [1], [0]]

x = solve(A,b)
x

```

```
---------------------------------------------------------------------------
LinAlgError                                   Tr
<ipython-input-25-6f9f965296ce> in <module>
      4 b = [[2], [1], [0]]
      5
----> 6 x = solve(A,b)
      7 x

1 frames
/usr/local/lib/python3.7/dist-packages/scipy
     27                              '.'.format(
     28     elif 0 < info:
---> 29         raise LinAlgError('Matrix is
     30
     31     if lamch is None:

LinAlgError: Matrix is singular.
```

```
[30]

# importing library sympy
from sympy import symbols, Eq, solve


x, y, z = symbols('x,y,z')

# defining equations
eq1 = Eq((x+y+z), 2)
eq2 = Eq((y-3*z), 1)
eq3 = Eq((2*x+y+5*z), 0)


print(solve((eq1, eq2, eq3), (x, y, z)))
```

```
[]
```

Matrix

- many solution

[31]

```python

# importing library sympy
from sympy import symbols, Eq, solve


x, y, z = symbols('x,y,z')

# defining equations
eq1 = Eq((-3*x - 5* y+ 36 * z), 10)
eq2 = Eq((-x + 7*z), 5)
eq3 = Eq((x+y-10*z), -4)


print(solve((eq1, eq2, eq3), (x, y, z)))
```

```
{x: 7*z - 5, y: 3*z + 1}
```

SJSU SAN JOSÉ STATE UNIVERSITY

Matrix

- Solving linear equations using Gaussian-Jordan elimination process
  - Convert the matrix to a echelon form

$$x + 2t - z = 10$$
$$-2x + z = 15$$
$$x - y = 20$$

Matrix

- Solving linear equations using Gaussian-Jordan elimination process
  - Convert the matrix to a echelon form

$$x + 2t - z = 10$$
$$-2x + z = 15$$
$$x - y = 20$$

```
[52]   1 A

array([[ 1,  2, -1],
       [-2,  0,  1],
       [ 1, -1,  0]])
```

```
[56]   1 y = np.array([10,15,20])
       2 y

array([10, 15, 20])
```

```
[58]   1 B = np.linalg.inv(A)
       2 B

array([[1., 1., 2.],
       [1., 1., 1.],
       [2., 3., 4.]])
```

```
       1 w = B@y
       2 w

array([ 65.,  45., 145.])
```

Matrix

- Solving linear equations using Gaussian-Jordan elimination process
  - Convert the matrix to a echelon form

$$x + 2t - z = 10$$
$$-2x + z = 15$$
$$x - y = 20$$

[52]
```
A
```

```
array([[ 1,  2, -1],
       [-2,  0,  1],
       [ 1, -1,  0]])
```

[56]
```
y = np.array([10,15,20])
y
```

```
array([10, 15, 20])
```

[60]
```
np.linalg.solve(A,y)
```

```
array([ 65.,  45., 145.])
```

Matrix

- Trace
  - A trace of a square matrix is the sum of the values on the main diagonal of the matrix

$$tr(A) = a_{1,1} + a_{2,2} + a_{3,3}$$

Matrix

- Determinant
  - The determinant of a square matrix is a scalar representation of the volume of the matrix

$$\det\begin{pmatrix} a & b \\ c & d \end{pmatrix} = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc.$$

Matrix

- Rank of a Matrix – dimension of the vector space generated (spanned) by its columns
  - The maximal number of linearly independent columns in the matrix

Reduced row echelon form

$$A = \begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 1 \\ -2 & -3 & 1 \\ 3 & 5 & 0 \end{bmatrix} \xrightarrow{2R_1+R_2\to R_2} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 3 & 5 & 0 \end{bmatrix} \xrightarrow{-3R_1+R_3\to R_3} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & -1 & -3 \end{bmatrix}$$

$$\xrightarrow{R_2+R_3\to R_3} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 0 \end{bmatrix} \xrightarrow{-2R_2+R_1\to R_1} \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & 3 \\ 0 & 0 & 0 \end{bmatrix}.$$

Matrix Decomposition/Factorization

- Matrix factorization - Factorization of a matrix into product of matrices
- LU factorization: $A = LU$; where L is a lower triangular and U is upper triangular matrix

$$L = \begin{bmatrix} \ell_{1,1} & & & & 0 \\ \ell_{2,1} & \ell_{2,2} & & & \\ \ell_{3,1} & \ell_{3,2} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ \ell_{n,1} & \ell_{n,2} & \ldots & \ell_{n,n-1} & \ell_{n,n} \end{bmatrix}$$

$$U = \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & \ldots & u_{1,n} \\ & u_{2,2} & u_{2,3} & \ldots & u_{2,n} \\ & & \ddots & \ddots & \vdots \\ & & & \ddots & u_{n-1,n} \\ 0 & & & & u_{n,n} \end{bmatrix}$$

LU decomposition is useful as it is computationally cheaper:

- computing inverse of upper or lower triangular matrix is easier ➔ solving linear systems faster

https://www.cfm.brown.edu/people/dobrush/cs52/Mathematica/Part2/PLU.html

Matrix Decomposition/Factorization

Solve system of linear equations ($\mathbf{Ax = b}$) using LU decomposition

Assume, $\mathrm{A} = LU$ and substitute and solve the system for x: $LUx = b$;

Let $Ux = y \Rightarrow Ly = b$

1. Solve $Ly = b \ for \ y$
2. Solve $Ux = y \ for \ x$

Example: in class

Matrix Decomposition/Factorization

```
from scipy.linalg import lu
A = np.array([[1,1,-1],[1,-2,3],[2,3,1]])
p, l, u = lu(A)
```

```
[149]  p
```

```
array([[0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

```
[150]  l
```

```
array([[1.        , 0.        , 0.        ],
       [0.5       , 1.        , 0.        ],
       [0.5       , 0.14285714, 1.        ]])
```

```
[151]  u
```

```
array([[ 2.        ,  3.        ,  1.        ],
       [ 0.        , -3.5       ,  2.5       ],
       [ 0.        ,  0.        , -1.85714286]])
```

```
[148]  np.allclose(A - p @ l @ u, np.zeros((3, 3)))
```

```
True
```

SJSU SAN JOSÉ STATE UNIVERSITY

Eigen values and Eigen Vectors

- An eigenvector of a $n \times n$ square matrix A is a **non-zero** vector $v \in R^n$ such that $Av = \lambda v$ for some scaler λ
- An eigenvalue of a $n \times n$ square matrix A is a scaler such that $Av = \lambda v$ has a non-trivial solution
  - Eigenvalues may be equal to zero

We do not consider the zero vector to be an eigenvector:
since A0=0=λ0 for *every* s
calar λ, the associated eigenvalue would be undefined.

$Av = \lambda v$ tells that $Av \ and \ \lambda v$ are collinear

➔$Av \ and \ v$ lie on the same line through the origin

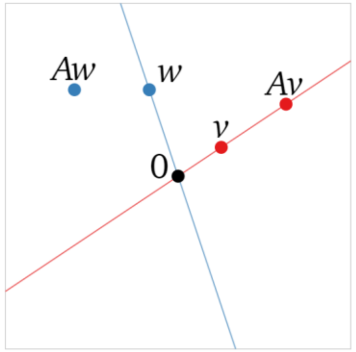
➔Av is a scaler multiple of v

https://textbooks.math.gatech.edu/ila/eigenvectors.html

Eigen values and Eigen Vectors

- An eigenvector of a $n \times n$ square matrix A is a non-zero vector $v \in R^n$ such that $Av = \lambda v$ for some scaler λ
- An eigenvalue of a $n \times n$ square matrix A is a scaler such that $Av = \lambda v$ has a non-trivial solution

We do not consider the zero vector to be an eigenvector: since A0=0=λ0 for *every* scalar λ, the associated eigenvalue would be undefined.



v is an eigen vector; however, w is not an eigenvector

$Av = \lambda v$ tells that Av *and* λv are collinear

➔ Av *and* v lie on the same line through the origin

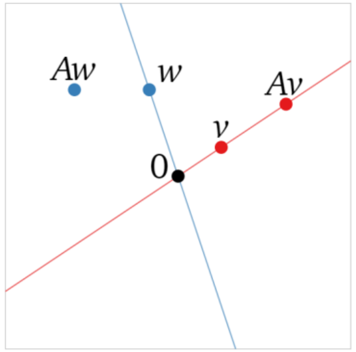
➔ Av is a scaler multiple of v

https://textbooks.math.gatech.edu/ila/eigenvectors.html

Eigen values and Eigen Vectors

Q: verify eigenvectors

$A = \begin{pmatrix} 1 & 1 \\ -2 & 4 \end{pmatrix}$ and vectors $v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ $\quad \mathrm{w} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$



v is an eigen vector; however, w is not an eigenvector

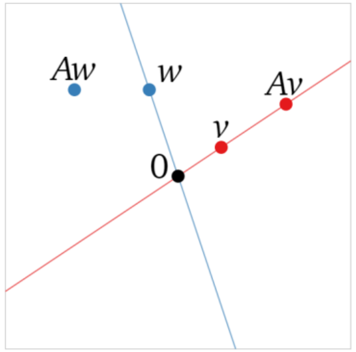
https://textbooks.math.gatech.edu/ila/eigenvectors.html

Eigen values and Eigen Vectors

$$A = \begin{pmatrix} 0 & 6 & 8 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{pmatrix} \quad \text{and vectors} \quad v = \begin{pmatrix} 16 \\ 4 \\ 1 \end{pmatrix} \quad w = \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix}$$

$$Av = \begin{pmatrix} 0 & 6 & 8 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} 16 \\ 4 \\ 1 \end{pmatrix} = \begin{pmatrix} 32 \\ 8 \\ 2 \end{pmatrix}$$

$$Aw = \begin{pmatrix} 0 & 6 & 8 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 28 \\ 1 \\ 1 \end{pmatrix}$$



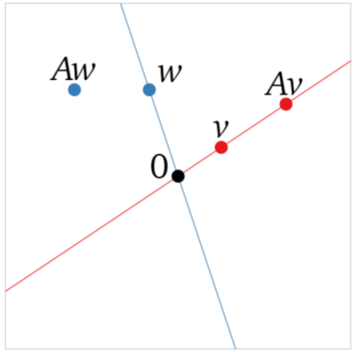
v is an eigen vector; however, w is not an eigenvector

https://textbooks.math.gatech.edu/ila/eigenvectors.html

Eigen values and Eigen Vectors

$$A = \begin{pmatrix} 1 & 3 \\ 2 & 6 \end{pmatrix} \qquad v = \begin{pmatrix} -3 \\ 1 \end{pmatrix}$$

$$Av = ?$$



v is an eigen vector; however, w is not an eigenvector

Eigen values and Eigen Vectors

- An eigenvector of a $n \times n$ square matrix A is a **non-zero** vector $v \in R^n$ such that $Av = \lambda v$ for some scaler λ
- An eigenvalue of a $n \times n$ square matrix A is a scaler such that $Av = \lambda v$ has a non-trivial solution
  - Eigenvalues may be equal to zero

Find eigenvectors and eigenvalues of a Matrix.
We can rewrite the equation:

$$\begin{aligned} & Av = \lambda v \\ \iff \quad & Av - \lambda v = 0 \\ \iff \quad & Av - \lambda I_n v = 0 \\ \iff \quad & (A - \lambda I_n)v = 0 \end{aligned}$$

Step1: Find the determinant of $(A - \lambda I)$ and solve it for zero vector
Step2: For each value of λ, find the associated vector

https://textbooks.math.gatech.edu/ila/eigenvectors.html

Eigen values and Eigen Vectors

- An eigenvector of a $n \times n$ square matrix A is a **non-zero** vector $v \in R^n$ such that $Av = \lambda v$ for some scaler λ
- An eigenvalue of a $n \times n$ square matrix A is a scaler such that $Av = \lambda v$ has a non-trivial solution
  - Eigenvalues may be equal to zero

Find eigenvectors and eigenvalues of a Matrix.
We can rewrite the equation:

$$
\begin{aligned}
& Av = \lambda v \\
\iff \quad & Av - \lambda v = 0 \\
\iff \quad & Av - \lambda I_n v = 0 \\
\iff \quad & (A - \lambda I_n)v = 0
\end{aligned}
$$

Step1: Find the determinant of $(A - \lambda I)$ and solve it for zero vector
Step2: For each value of λ, find the associated vector
- Replace the value of λ in the characteristic polynomial equation

$$A = \begin{pmatrix} 1 & 1 \\ -2 & 4 \end{pmatrix}$$

Find eigenvectors and eigenvalues.

https://textbooks.math.gatech.edu/ila/eigenvectors.html

Eigen values and Eigen Vectors

[113]
```python
A = np.array([[1,1],[-2,4]])
A
```
```
array([[ 1,  1],
       [-2,  4]])
```

[114]
```python
evals, evecs = np.linalg.eig(A)
```

[115]
```python
evals
```
```
array([2., 3.])
```

[116]
```python
evecs
```
```
array([[-0.70710678, -0.4472136 ],
       [-0.70710678, -0.89442719]])
```

[117]
```python
v1 = evecs[:,0]
v2 = evecs[:,1]
```

[118]
```python
A@v1
```
```
array([-1.41421356, -1.41421356])
```

[119]
```python
evals[0] * v1
```
```
array([-1.41421356, -1.41421356])
```

[126]
```python
v3 = np.array([10,10])
v3
```
```
array([10, 10])
```

```python
A@v3
```
```
array([20, 20])
```

[125]
```python
evals[0] * v3
```
```
array([20., 20.])
```

$$A = \begin{pmatrix} 1 & 1 \\ -2 & 4 \end{pmatrix}$$

Find eigenvectors and eigenvalues.

Principal Component Analysis (PCA)

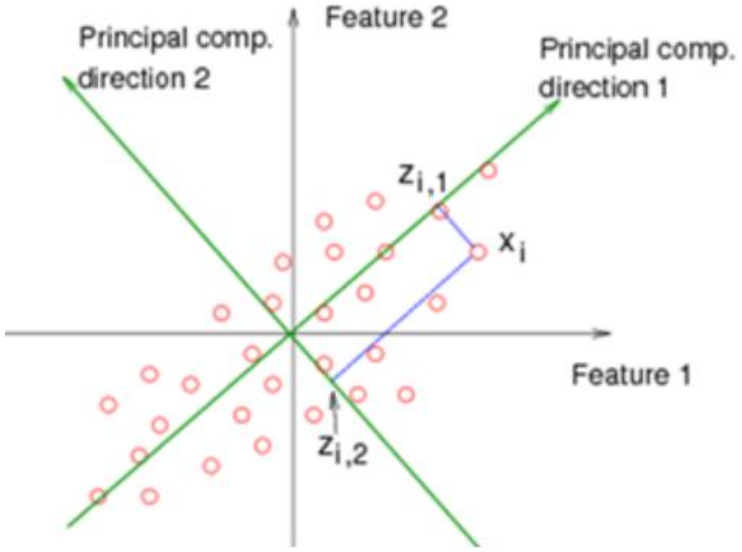
- PCA is a technique to simply a dataset
- PCA uses a linear transformation that transform the given dataset into a new coordinate system such that
  - The first principal component (PC1) will capture the maximum variance of the dataset
  - The PC2 will capture the second maximum variance of the dataset and so on
- PCA can be used for dimensionality reduction

https://en.wikipedia.org/wiki/Covariance

SJSU SAN JOSÉ STATE UNIVERSITY

Principal Component Analysis (PCA)

- PCA is a technique to simply a dataset
- PCA uses a linear transformation that transform the given dataset into a new coordinate system such that
  – The first principal component (PC1) will capture the maximum variance of the dataset
  – The PC2 will capture the second maximum variance of the dataset and so on
- PCA can be used for dimensionality reduction



https://online.stat.psu.edu/stat857

Principal Component Analysis (PCA)

- PCA is a technique to simply a dataset
- PCA uses a linear transformation that transform the given dataset into a new coordinate system such that
  - The first principal component (PC1) will capture the maximum variance of the dataset
  - The PC2 will capture the second maximum variance of the dataset and so on
- PCA can be used for dimensionality reduction

Why dimensionality reduction is important?

1. Reduce time and storage to train a model
2. Addressing multi-collinearity issue
   1. Improve model's parameter interpretation
3. Data visualization
4. Avoid curse of dimensionality

https://en.wikipedia.org/wiki/Covariance

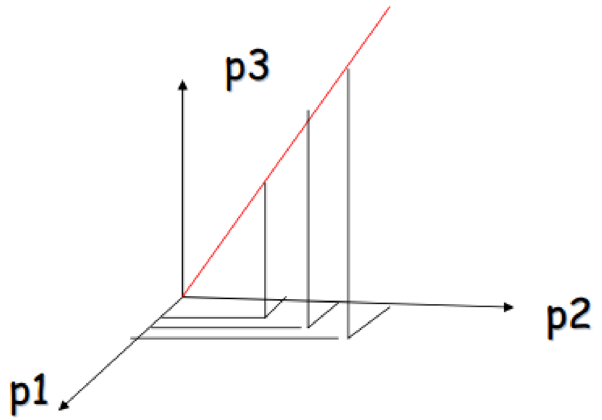
Principal Component Analysis (PCA)

- Consider the points in 3D: [(1, 2, 3), (2,4,6), (-1, -2, -3), (10,20, 30), (-5, - 10, -15)]
  – If each elements need one byte for storing, total bytes?
  – Can we reduce the storage size?
    - Notice the points

Image credit-https://www.cse.psu.edu/~rtc12/CSE586Spring2010/lectures/pcaLectureShort.pdf

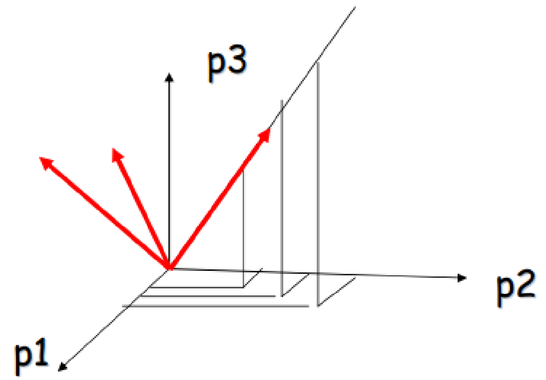
Principal Component Analysis (PCA)

- Consider the points in 3D: [(1, 2, 3), (2,4,6), (3, 6, 9), (10,20, 30), (5, 10, 15)]
  - If each elements need one byte for storing, total bytes?
  - Can we reduce the storage size?
    - Notice the points



All the points fall on a line (1D)

New coordinate system where one of the axes is along the direction of the line



Now, we need only the direction of the line (a 3 bytes image) and non-zero coordinate for each of the point

Image credit-https://www.cse.psu.edu/~rtc12/CSE586Spring2010/lectures/pcaLectureShort.pdf

Principal Component Analysis

- Covariance
  - a measure of spread of data points around the center of the mean
    - In multi-dimensional space, covariance measures how much each of the dimensions vary from the mean with respect to each other
    - In 2D: it measures the relationship between the two dimensions, e.g. number of hours study vs. GPA achieved
    - Covariance of one variable to itself is the variance
    - The normalized version of the covariance is the correlation coefficient

$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

$cov_{x,y}$ = covariance between variable x and y

x_i = data value of x

y_i = data value of y

$\bar{x}$ = mean of x

$\bar{y}$ = mean of y

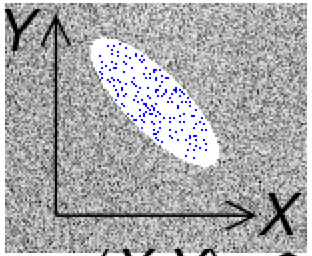
N = number of data values

Principal Component Analysis

- Diagonal are the variances
- Covariance matrix is symmetric
  - $cov(x, y) = cov(y, x)$
- n-dimension data will generate a $n \times n$ dimension of the covariance matrix
  - e.g., for 3-D data we have 3x3 dimensional covariance matrix

$$C = \begin{bmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{bmatrix}$$

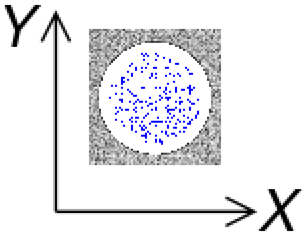
Principal Component Analysis



$\mathrm{cov}(X,Y)<0$

$$\begin{bmatrix} 2 & -3 \\ -3 & 1 \end{bmatrix}$$

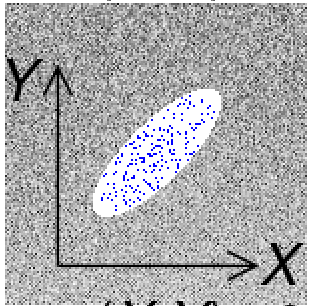
negative covariance



$\mathrm{cov}(X,Y)\approx 0$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1.5 \end{bmatrix}$$

zero covariance



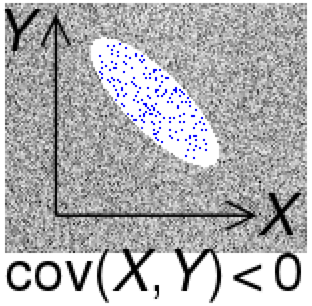
$\mathrm{cov}(X,Y)>0$

$$\begin{bmatrix} 1 & 2 \\ 2 & 1.5 \end{bmatrix}$$

Positive covariance

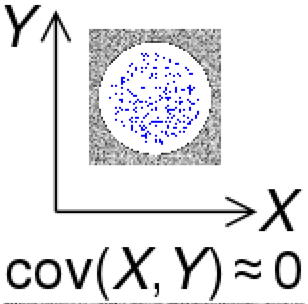
https://en.wikipedia.org/wiki/Covariance

Principal Component Analysis



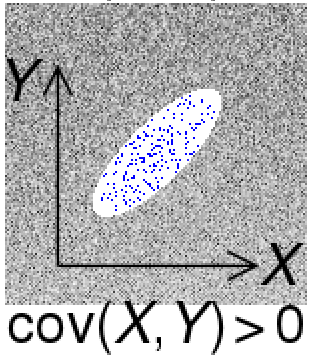
$$\begin{bmatrix} 2 & -3 \\ -3 & 1 \end{bmatrix}$$

negative covariance



$$\begin{bmatrix} 1 & 0 \\ 0 & 1.5 \end{bmatrix}$$

zero covariance



$$\begin{bmatrix} 1 & 2 \\ 2 & 1.5 \end{bmatrix}$$

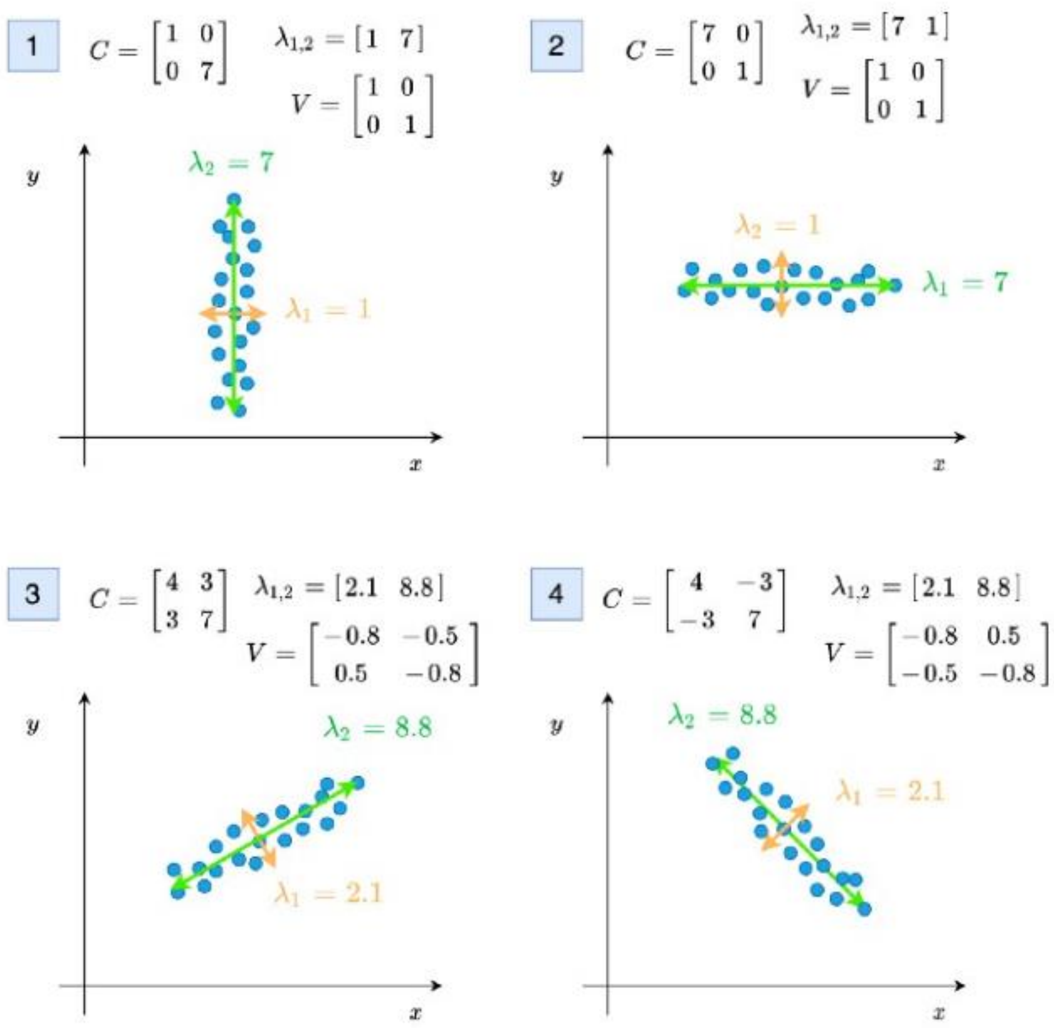
Positive covariance

We are also interested to find the magnitude and direction of a covariance matrix

We can use Eigen values and Eigen vector can for magnitude and direction

https://en.wikipedia.org/wiki/Covariance

Principal Component Analysis



We are also interested to find the magnitude and direction of a covariance matrix

We can use Eigen values and Eigen vector can for magnitude and direction

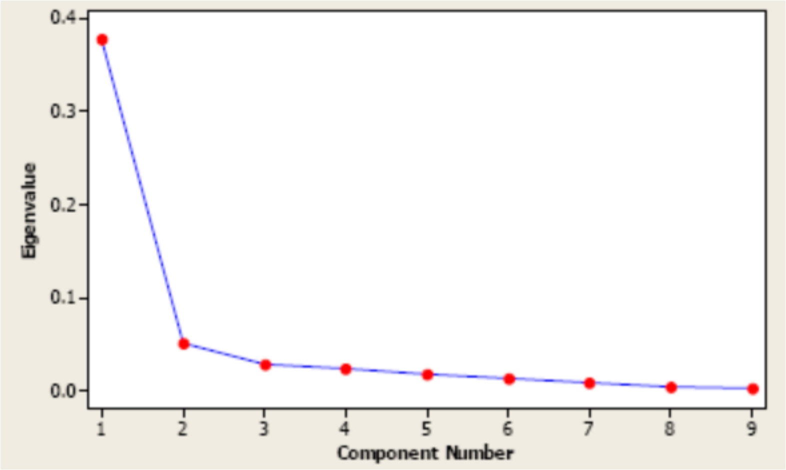
https://towardsdatascience.com/5-things-you-should-know-about-covariance-

Principal Component Analysis

Step 1: Find the mean centered data
Step 2: find covariance matrix of the mean centered data
Step 3: find the eigenvalues and associated eigen vectors
Step 4: sort the eigenvalues and associated eigenvectors
Step 5: Transform the original data and reduce the dimensionality as needed

Principal Component Analysis

- We can find eigenvalues and eigenvectors of the covariance matrix where the eigenvectors with the largest eigenvalues correspond to the dimensions that have the strongest correlation in the dataset



Explained variance by the principal components

Principal Component Analysis

- Iris Dataset

```python
from sklearn import datasets
iris = datasets.load_iris()
```

```python
data = iris['data'] # dropping the class variable
```

```python
[42]
# PCA
from sklearn.decomposition import PCA
```

```python
pca = PCA()
pca.fit(X)
X = pca.transform(data)
```

Data Set Characteristics:	Multivariate	**Number of Instances:**	150	**Area:**	Life
Attribute Characteristics:	Real	**Number of Attributes:**	4	**Date Donated**	1988-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	4899557

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Attributes are:

1. Sepal length
2. Sepal width
3. Petal length
4. Petal width
5. class:
   -- Iris Setosa
   -- Iris Versicolour
   -- Iris Virginica

https://archive.ics.uci.edu/ml/datasets/iris

Principal Component Analysis

- Iris Dataset

```python
from sklearn import datasets
iris = datasets.load_iris()
```

```python
data = iris['data'] # dropping the class variable
```

[42]

```python
# PCA
from sklearn.decomposition import PCA
```

```python
pca = PCA()
pca.fit(X)
X = pca.transform(data)
```

```python
pca.components_
```

```
array([[ 0.36138659, -0.08452251,  0.85667061,  0.3582892 ],
       [ 0.65658877,  0.73016143, -0.17337266, -0.07548102],
       [-0.58202985,  0.59791083,  0.07623608,  0.54583143],
       [-0.31548719,  0.3197231 ,  0.47983899, -0.75365743]])
```

```python
pca.explained_variance_
```

```
array([4.22824171, 0.24267075, 0.0782095 , 0.02383509])
```

```python
pca.explained_variance_ratio_
```

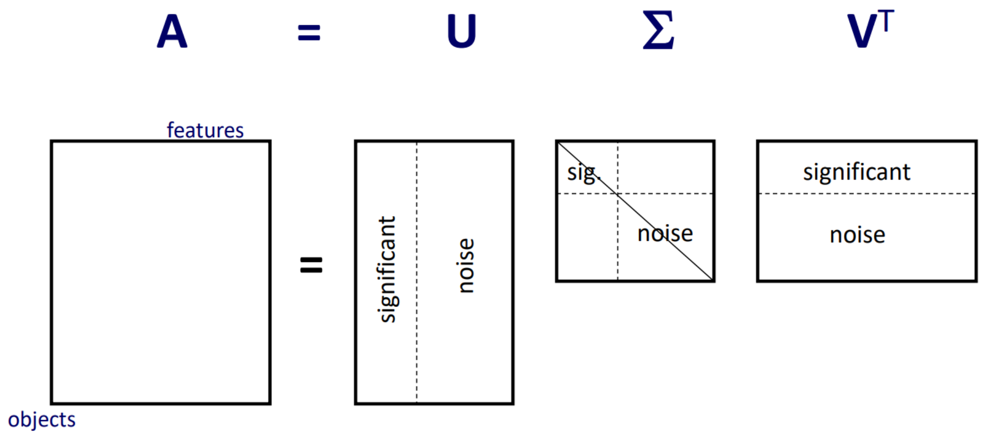
```
array([0.92461872, 0.05306648, 0.01710261, 0.00521218])
```

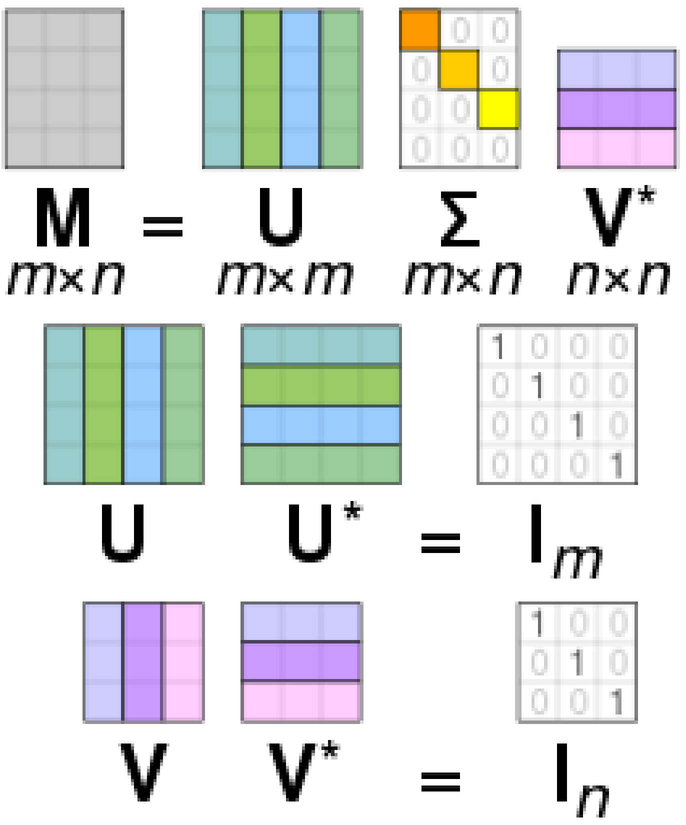
SJSU SAN JOSÉ STATE UNIVERSITY

Principal Component Analysis

- Limitations of PCA

Singular Value Decomposition(SVD)

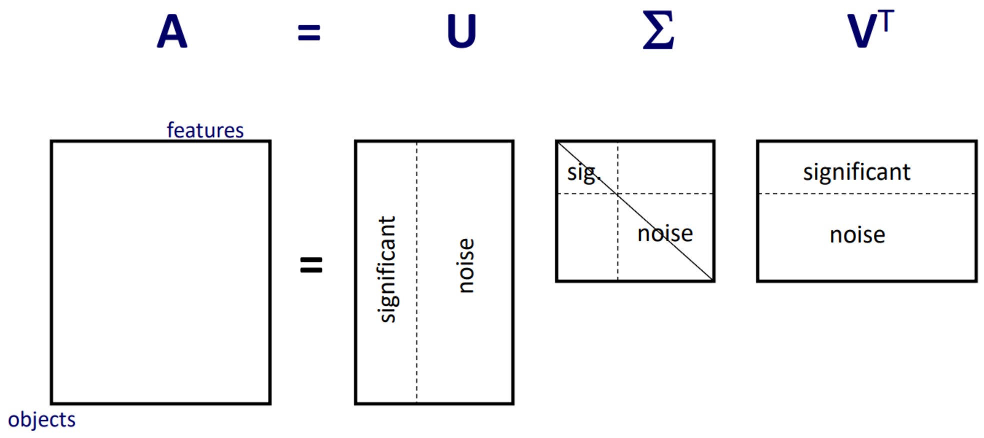


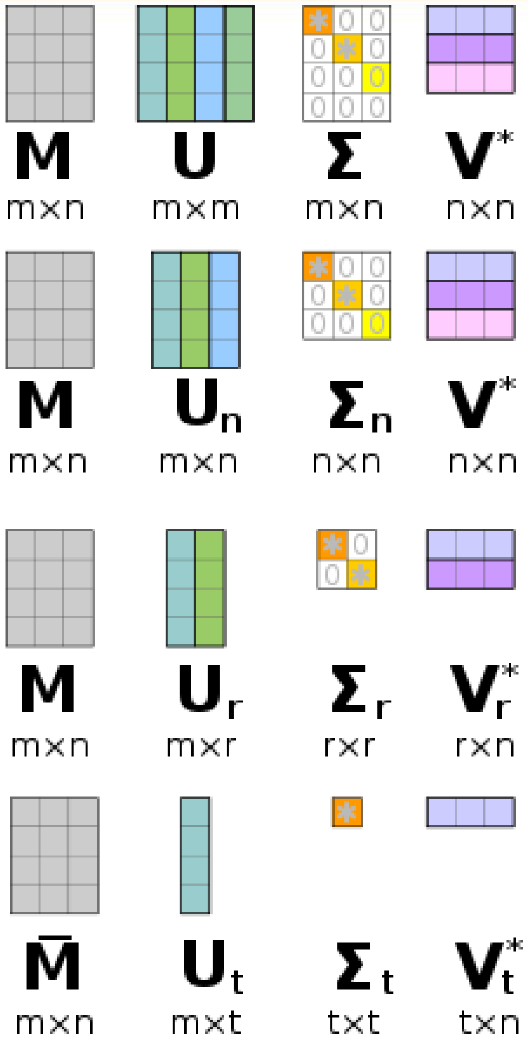


- Where A is the real m × n matrix that we want to decompose, U is an m × m matrix, Σ is an m × n diagonal matrix, and V T is the V transpose of an n × n matrix

https://en.wikipedia.org/wiki/Singular_value_decomposition

Singular Value Decomposition(SVD)





- Where A is the real m × n matrix that we want to decompose, U is an m × m matrix, Σ is an m × n diagonal matrix, and V T is the V transpose of an n × n matrix