Vector Math, Vector Embedding & Vector Database

Practical Foundations untuk Praktisi Machine Learning & AI

Wuriyanto

https://wuriyan.to

Apa itu Vector

Dari sudut pandang **Matematika** (*Math, Linear Algebra*) dan **Fisika** (*Physics*), definisi **Vector** adalah objek yang memiliki besar (*magnitude*) dan arah (*direction*).

Definisi sederhana: **Vector** adalah objek yang merepresentasikan suatu entitas menggunakan **lebih dari satu nilai**. Mengapa lebih dari satu nilai? **Sebab Entitas di dunia nyata bersifat Multidimensional (tidak hanya memiliki satu sifat saja)**.

Contoh:

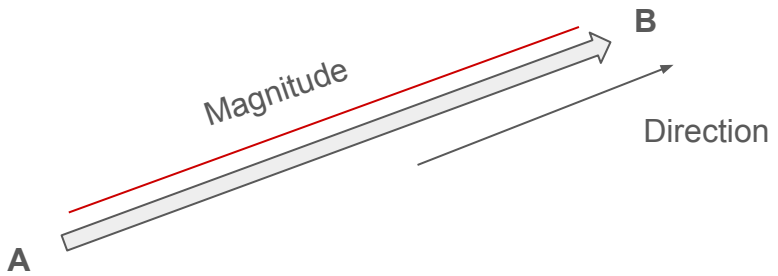
Manusia memiliki: tinggi badan, berat, umur | contoh format dalam Vector [180, 78, 26]

Warna (format RGB) biasanya 3 atau 4 dimensi : Red, Green, Blue, dan Alpha (untuk mengatur *opacity*) | contoh format dalam Vector [138, 26, 18, 0.5]

Vector memungkinkan:

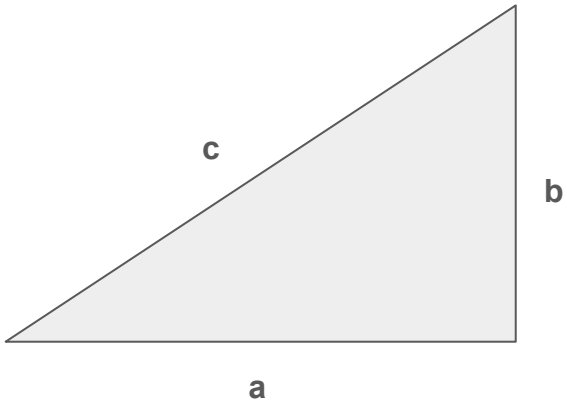
- setiap dimensi menyimpan **informasi berbeda**
- kombinasi dimensi membentuk identitas entitas

Secara Matematis: $v=(x1,x2,\ldots,xn)$



Teorema Pythagoras (Pythagorean Theorem)

Teorema Pythagoras menjelaskan hubungan antara panjang sisi pada segitiga siku-siku (*right angled triangle*).



$$c^2 = a^2 + b^2$$

$$c = \sqrt{a^2 + b^2}$$

Contoh:

$$c^2 = 4^2 + 3^2$$

$$c = \sqrt{4^2 + 3^2}$$

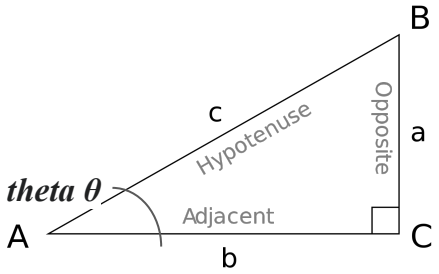
$$c = 5$$

Trigonometry

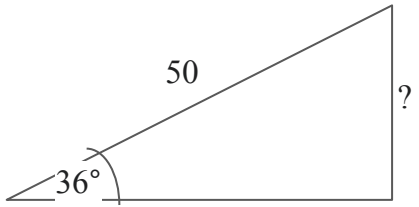
Trigonometry adalah cabang Matematika yang mempelajari hubungan sisi dan sudut segitiga menggunakan rasio seperti **sine (sin)**, **cosine (cos)**, **tangent (tan)** dan fungsi *inverse*-nya, seperti **Arcsine**, **Arccosine** dan **Arctangent**.

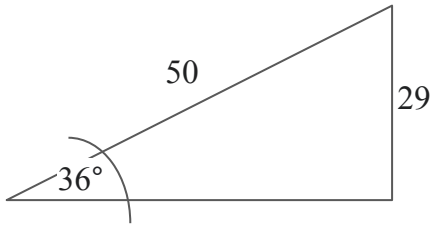
Trigonometry sangat penting bagi banyak industri. Sebut saja penerbangan, kelautan, otomasi mesin. Dan tentu saja sangat penting pada bidang Ilmu Komputer, seperti pada pengolahan citra digital, *Computer vision, AI, Machine Learning, Game engine*.

Kita tidak akan membahas detail tentang Trigonometri, hanya akan kita bahas sedikit beberapa persamaan yang akan kita gunakan sesuai tema materi ini.



$\sin\theta = a/c$
$\cos\theta = b/c$
$\tan\theta = a/b$





Kita akan coba buktikan beberapa persamaan diatas. Pada gambar disamping diketahui sudut kemiringan segitiga tersebut **θ = 36°**, nilai **hypotenuse atau c = 50** dan nilai yang ingin kita ketahui adalah **opposite atau a**. Sehingga kita bisa menggunakan $\sin(\theta) = a/c$.

sin(θ) = a/c

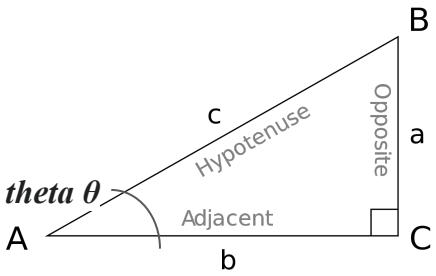
sin(36°) = a/50
a = sin(36°) . 50

sin(36°) = 0.58

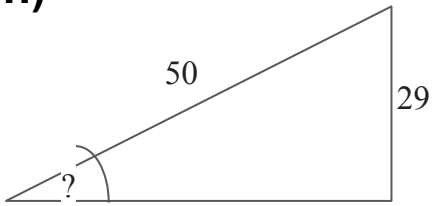
a = 0.58 x 50

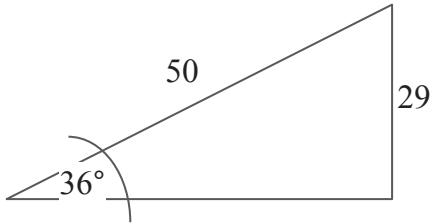
a = 29

Trigonometry (inverse function)



$$\sin\theta = a/c$$





function	*inverse function*
$\sin\theta = a/c$	$\theta = \text{arcsine}(a/c)$
$\cos\theta = b/c$	$\theta = \text{arccosine}(b/c)$
$\tan\theta = a/b$	$\theta = \text{arctangent}(a/b)$

Sama halnya, misal ketika kita ingin mengetahui berapa **sudut *θ (theta)*,** kita bisa memanfaatkan fungsi ***inverse*** trigonometri. Pada gambar disamping diketahui nilai **hypotenuse atau c = 50, opposite atau a = 29** dan **nilai yang ingin kita ketahui adalah sudut *θ (theta)***. Kita masih menggunakan persamaan $\sin\theta = a/c$.

sin(θ) = a/c

sin(θ) = 29/50

29/50 = 0.58

sin(θ) = 0.58

θ = arcsine(0.58)

θ = 36°

Scalar vs Vector

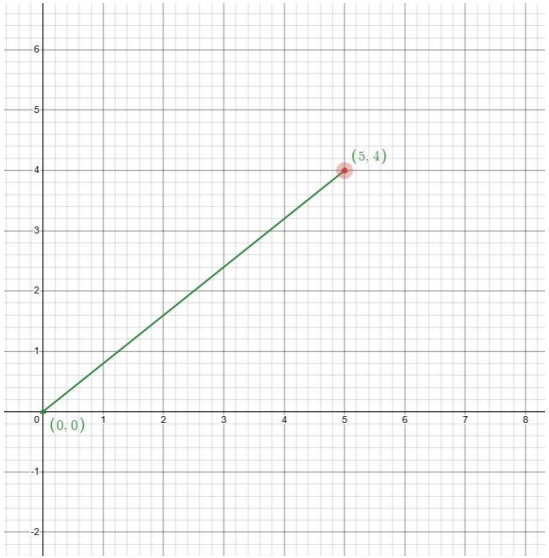
Dari sudut pandang **Matematika** (*Math*) dan **Fisika** (*Physics*), ***Scalar*** adalah objek yang hanya memiliki besaran (*magnitude*) saja. Berbeda dengan **Vector**, yang memiliki besaran (*magnitude*) dan arah (*direction*).

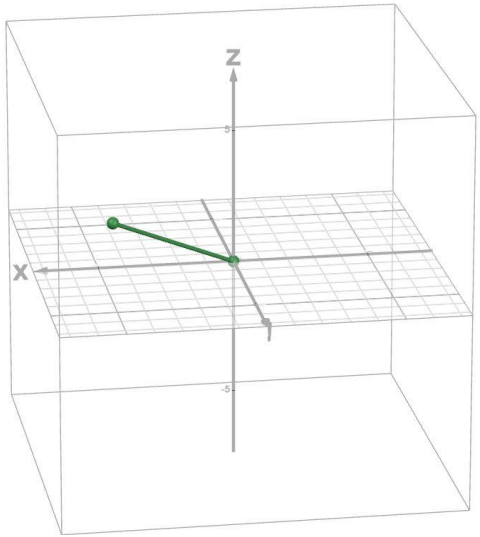
	Scalar	Vector
Jumlah Nilai	Satu	Banyak Nilai
Arah	Tidak punya arah	Punya arah
Contoh	Suhu (30°)	Akselerasi (5 km/h) ke arah timur

Representasi Vector

Secara fundamental **Vector** bisa direpresentasikan melalui sudut pandang **Geometri** (*Geometry*) dan **Aljabar** (*Algebra*).

Pada sudut pandang *Geometry*, **Vector** dianggap sebagai sebuah **panah**(*arrow*) yang ada pada ruang **2D**, **3D** dan **ND**.





Pada sudut pandang Aljabar (*Algebra*), Vector direpresentasikan dengan daftar angka yang berurut (*ordered list of number*).

V = (x, y)

V = (x, y, z)

V = (x, y, z, ….., N)

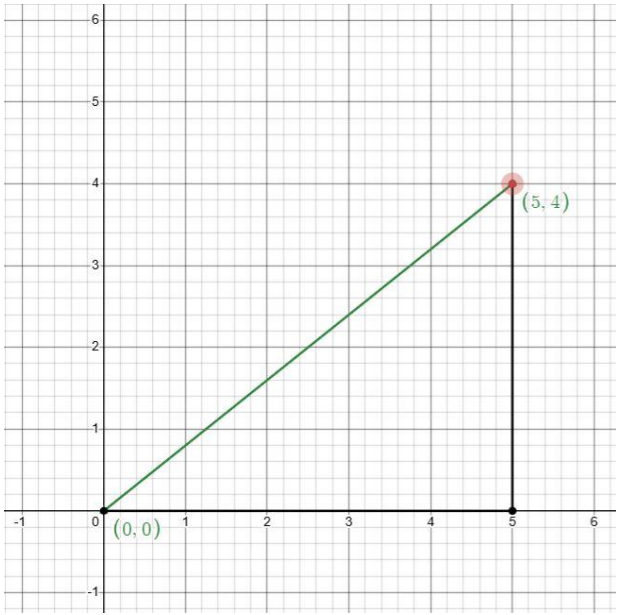
V = [5, 4]

V = [5, 4, 3]

V = [0.4, 0.66, 0.7, 1, …… 0.32]

Besaran/Panjang (Magnitude) Vector pada ruang 2 Dimensions

Untuk menghitung besaran/panjang (*Magnitude*) dari sebuah Vector, kita perlu mencari **akar kuadrat** dari penjumlahan dari setiap komponen Vector yang sudah di **kuadratkan**.



Pada ruang **2 Dimensi** seperti disamping, Vector memiliki 2 komponen, yaitu x dan y. Atau pada pembahasan **Representasi Vector**, bisa kita tuliskan dengan:

x = 5

y = 4

V = [5, 4]

Jika kita perhatikan, Vector dan komponen-komponennya membentuk Segitiga siku-siku. Sehingga pada dasarnya kita bisa menghitung besar/panjang (*Magnitude*) Vector dengan formula Teorema Pythagoras (*Pythagorean Theorem*).

$$c^2 = a^2 + b^2$$

$$c = \sqrt{a^2 + b^2}$$

$$\|m\| = \sqrt{x^2 + y^2}$$

$$\|m\| = \sqrt{5^2 + 4^2}$$

$$\|m\| = 6.4$$

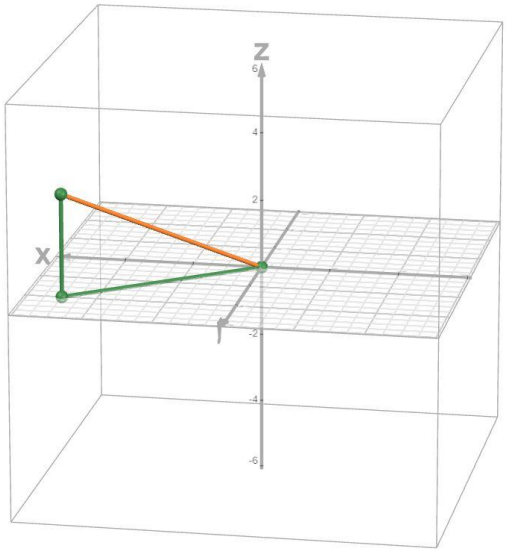
Sehingga Vector:

V = [5,4]

Memiliki besaran/panjang (Magnitude) = ~6.4

Besaran/Panjang (Magnitude) Vector pada ruang 3 Dimensions

Untuk menghitung besaran/panjang (*Magnitude*) dari sebuah Vector pada ruang 3D kita tetap bisa menggunakan *Pythagorean Theorem*.



Pada ruang **3 Dimensi** seperti disamping, Vector memiliki 3 komponen, yaitu x, y dan z. Atau pada pembahasan Representasi Vector, bisa kita tuliskan dengan:

x = 5

y = 4

Z = 3

V = [5, 4, 3]

Jika kita perhatikan, Vector dan komponen-komponennya membentuk Segitiga siku-siku. Sehingga pada dasarnya kita bisa menghitung besar/panjang (*Magnitude*) Vector dengan formula Teorema Pythagoras (*Pythagorean Theorem*).

$$d^2 = a^2 + b^2 + c^2$$

$$d = \sqrt{a^2 + b^2 + c^2}$$

$$\|m\| = \sqrt{x^2 + y^2 + z^2}$$

$$\|m\| = \sqrt{5^2 + 4^2 + 3^2}$$

$$\|m\| = 7$$

Sehingga Vector:

V = [5,4,3]

Memiliki besaran/panjang (*Magnitude*) = ~7

Besaran/Panjang (Magnitude) Vector pada ruang N Dimensions

Untuk menghitung besaran/panjang (*Magnitude*) dari sebuah Vector pada ruang ND kita tetap bisa menggunakan *Pythagorean Theorem*.

V = [x1, x2, x3, …… xn]

Bentuk General:

$$||m|| = \sqrt{x1^2 + x2^2 + x3^2 .....xn^2}$$

$$||m|| = \sqrt{\sum_{i=1}^{n} di^2}$$

Unit Vector dan Vector Normalization

Unit Vector Part 1

$$\|\hat{v}\| = 1$$

Unit Vector adalah Vector yang memiliki besaran/panjang (*magnitude*) = 1.

Untuk mendapatkan Unit Vector, lakukan pembagian setiap elemen Vector-nya dengan magnitude-nya. **Proses ini disebut juga dengan Vector Normalization.**

Vector source:

v = [v1, v2, v3, … vn]

$$\hat{v} = \frac{v}{\|v\|}$$

dimana:

$\hat{v}$ Unit Vector

v Vector Source

$\|v\|$ Vector Magnitude

$$\|v\| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

Bagi setiap elemen pada Vector source dengan ***magnitude*-nya**:

$$\hat{v} = [\frac{v_1}{\|v\|}, \frac{v_2}{\|v\|}, \frac{v_3}{\|v\|}, \ldots, \frac{v_n}{\|v\|}]$$

Unit Vector Part 2

Contoh Mendapatkan Unit Vector

Vector source:

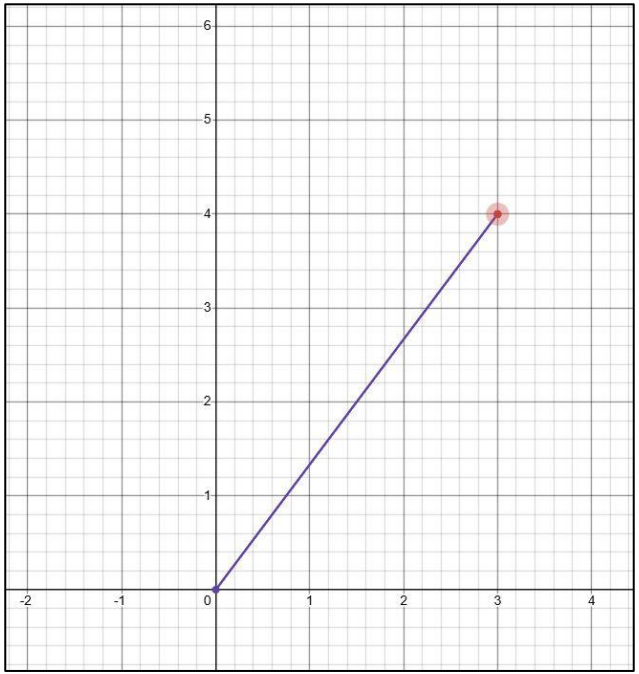
v = [3, 4]

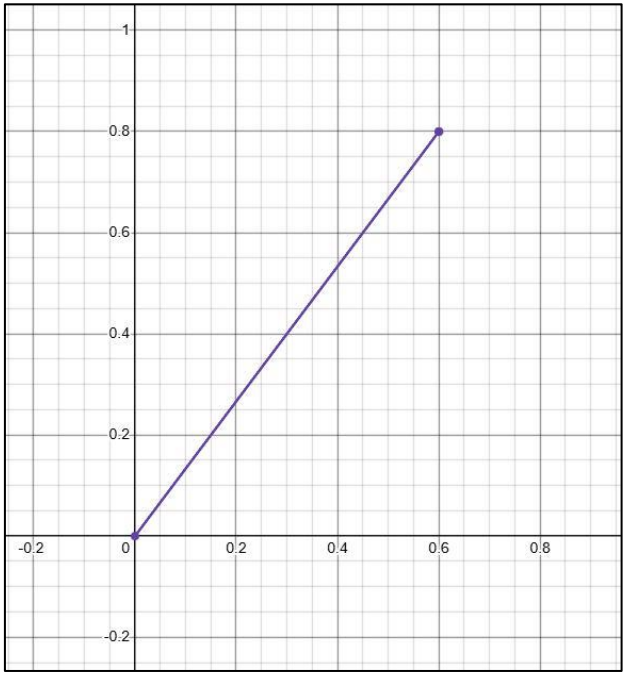
$$\|v\| = \sqrt{3^2 + 4^2}$$

$$\|v\| = 5$$

$$\hat{v} = [\frac{3}{5}, \frac{4}{5}]$$

$$\hat{v} = [0.6, 0.8]$$





Unit Vector Part 3

Contoh Mendapatkan Unit Vector

Pada gambar dibawah, **Vector Source (kiri)**, dan **Unit Vector-nya (Kanan) arah (direction) tidak berubah**, meskipun **Vector sudah di normalisasi ke dalam Unit Vector**.

Vector source:

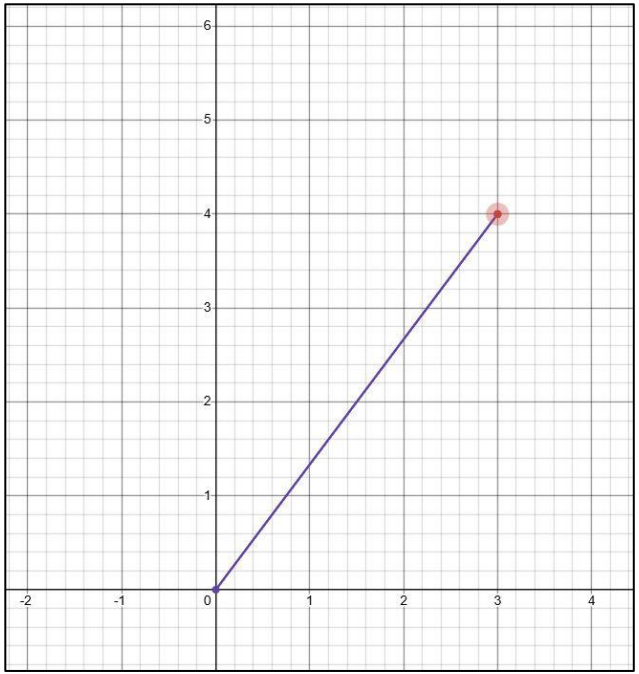
v = [3, 4]

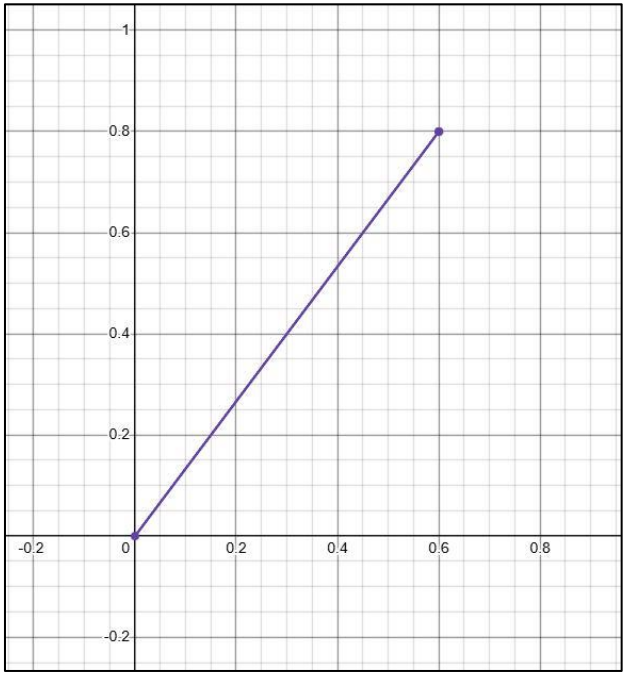
$$\|v\| = \sqrt{3^2 + 4^2}$$

$$\|v\| = 5$$

$$\hat{v} = [\frac{3}{5}, \frac{4}{5}]$$

$$\hat{v} = [0.6, 0.8]$$





Unit Vector Part 4

Makna intutif Unit Vector

- Unit Vector hanya merepresentasikan arah (*direction*)
- Fokus pada arah (*direction*) bukan besar/panjang (*magnitude*)
- Besarnya sudah dinormalisasi (dibagi dengan magnitude-nya)

Unit Vector Part 5

Mengapa Unit Vector penting?

- Mengisolasi arah (*direction*), panjang dihilangkan dan arah tetap (arah lebih penting).
- Perbandingan adil, dalam artian: Vector lebih besar bukan berarti lebih penting(bobot lebih tinggi).
- Unit Vector memungkinkan perbandingan makna tanpa dipengaruhi oleh besarnya nilai.

  Contoh kasus NLP (Natural Language Processing).

  A = “Saya suka Kopi”

  B = “Saya suka Kopi, tapi Kopi Arabika

  Setelah melalui proses **Embedding**, misalnya menggunakan model *modern* seperti **Sentence Transformers** atau **OpenAi Embedding** akan menghasilkan r**epresentasi Vector** dari dua kalimat diatas.

  A = [0.21, 0.30, 0.12, 0.40, ...]

  B = [0.42, 0.58, 0.25, 0.80, ...]

  Observasi:

- Kalimat A dan B memiliki **dimensi vector yang sama(Sentence Transformers: 384 / 768, Open Ai Embedding: 1536 / 3072).** Namun, **magnitude vector bisa berbeda.**
- Jika dibandingkan langsung: Dot Product besar, B terlihat lebih penting. Padahal secara topik sama, yaitu tentang “Kopi”.
- Perbedaan ini dipengaruhi oleh:
  - spesifikasi makna
  - intensitas representasi semantik. Pada kalimat B kata “Kopi” banyak dipasangkan dengan kata penting lain, misalnya “Arabika”. Sehingga banyak value Embedding lebih besar di dimensi tertentu.

Unit vector memastikan perbandingan dilakukan pada makna, bukan pada kekuatan aktivasi embedding.

Sparse Vector dan Dense Vector

Sparse Vector dan Dense Vector Part 1

Sparse Vector

Sparse Vector adalah Vector yang sebagian besar elemennya bernilai **nol**. Hanya sedikit dimensi yang memiliki nilai tidak nol.

- **Karakteristik:** Memiliki dimensi yang sangat besar (bisa ribuan hingga jutaan), namun sangat sedikit data aktif di dalamnya.
- **Contoh Representasi:** *Bag-of-Words* (BoW) atau *TF-IDF*.
- **Cara Kerja:** Jika kita memiliki kamus berisi 100.000 kata unik, dan satu kalimat hanya berisi 5 kata, maka Vector untuk kalimat tersebut akan memiliki 99.995 elemen bernilai nol dan hanya 5 elemen yang memiliki nilai.

Kelebihan & Kekurangan Sparse Vector

- **Kelebihan:** Sangat mudah diinterpretasikan (setiap dimensi mewakili kata tertentu secara eksplisit).
- **Kekurangan:** Membutuhkan ruang penyimpanan yang besar jika tidak dikompresi, dan tidak bisa menangkap hubungan makna (sinonim) antar kata.

Sparse Vector dan Dense Vector Part 2

Contoh Sparse Vector (TF-IDF/One-Hot)

Misalkan kita memiliki **"Kosakata" (Vocabulary)** yang terdiri dari 5 kata: `["apel", "buku", "ceri", "durian", "es"]`

Bayangkan kita ingin merepresentasikan kalimat: **"Apel dan Ceri"**.

Karena kata "dan" tidak ada dalam kamus kita, maka Vector-nya hanya akan melihat kata yang tersedia. Vector ini akan menandai 1 untuk kata yang muncul dan 0 untuk yang tidak.

Vector-nya:

[1, 0, 1, 0, 0]

- **Indeks 0 (apel):** 1 (Muncul)
- **Indeks 1 (buku):** 0 (Tidak muncul)
- **Indeks 2 (ceri):** 1 (Muncul)
- **Indeks 3 (durian):** 0 (Tidak muncul)
- **Indeks 4 (es):** 0 (Tidak muncul)

Dalam data yang nyata (misal jutaan kata), deretan angka 0 ini akan sangat panjang, itulah sebabnya disebut **Sparse** (jarang).

Sparse Vector dan Dense Vector Part 3

Dense Vector

Dense Vector adalah Vector yang sebagian besar atau seluruh elemennya memiliki nilai **bukan nol**. Biasanya nilainya berupa angka desimal (*floating point*).

- **Karakteristik:** Memiliki dimensi yang jauh lebih kecil dan tetap (misalnya 128, 256, atau 768 dimensi), namun setiap dimensi mengandung informasi "padat".
- **Contoh Representasi:** *Word Embeddings* (Word2Vec, GloVe) atau *Neural Embeddings* (BERT, OpenAI Embeddings).
- **Cara Kerja:** Alih-alih memetakan kata ke indeks kamus, dense vector memetakan data ke dalam ruang Vector kontinu di mana posisi Vector tersebut ditentukan oleh konteks dan makna.

Kelebihan & Kekurangan Dense Vector

- **Kelebihan:** Mampu menangkap hubungan semantik. Misalnya, dalam ruang dense vector, kata "Raja" dan "Ratu" akan berada di posisi yang berdekatan.
- **Kekurangan:** Sulit diinterpretasikan secara langsung oleh manusia (kita tidak tahu apa arti spesifik dari dimensi ke-45 dalam sebuah embedding).

Sparse Vector dan Dense Vector Part 4

Contoh Dense Vector

Dalam ***Dense Vector***, kata tidak lagi diwakili oleh posisi indeks di kamus, melainkan oleh sekumpulan angka desimal yang mewakili "makna" atau fitur tertentu dalam ruang multidimensi.

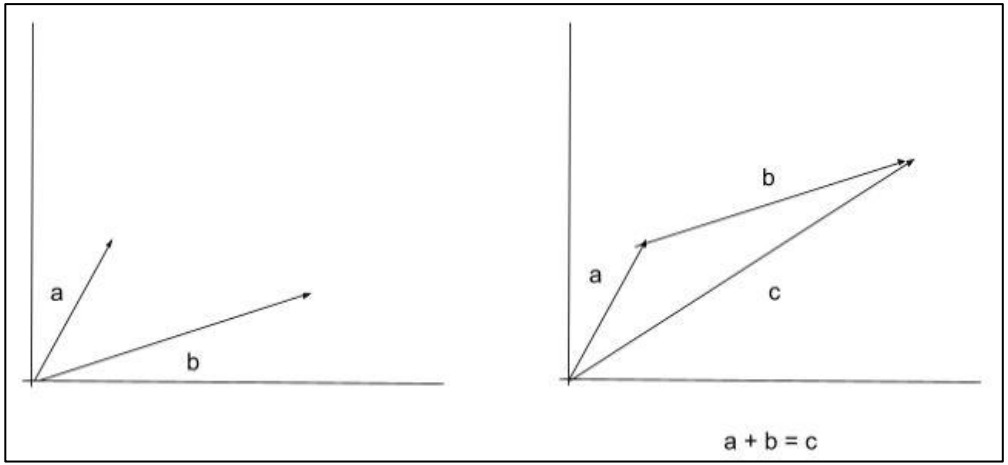
Misalnya, kata **"Apel"** dalam model AI seperti Sentence Transformers dan OpenAi Embedding mungkin direpresentasikan seperti ini (biasanya panjangnya 384, 768, atau lebih):

Vector-nya:

[0.12, -0.59, 0.88, 0.01, -0.34, ...]

- Setiap angka di atas tidak mewakili kata spesifik, melainkan fitur abstrak (seperti tingkat "kemanisan", "kemerahan", atau "kategori buah").
- Semua posisi terisi oleh angka (tidak ada nol yang dominan), itulah sebabnya disebut **Dense** (padat).

Operasi Vector | Penjumlahan Vector (Vector Addition) Part 1



Menjumlahkan 2 buah Vector cukup mudah. Dengan catatan 2 Vector tersebut mempunyai dimensi yang sama. Seperti gambar di samping, Vector a ditambahkan dengan Vector b akan menghasilkan Vector c. **Menjumlahkan satu Vector dengan Vector lainnya akan merubah *magnitude* dan *direction* dari Vector tersebut**.

Operasi Vector | Penjumlahan Vector (Vector Addition) Part 2

Contoh:

a = [3, 4]

b = [2, − 1]

Vector c berwarna ***orange*** adalah hasil penjumlahan Vector a dan vector b.

$c.x = a.x + b.x$

$c.y = a.y + b.y$

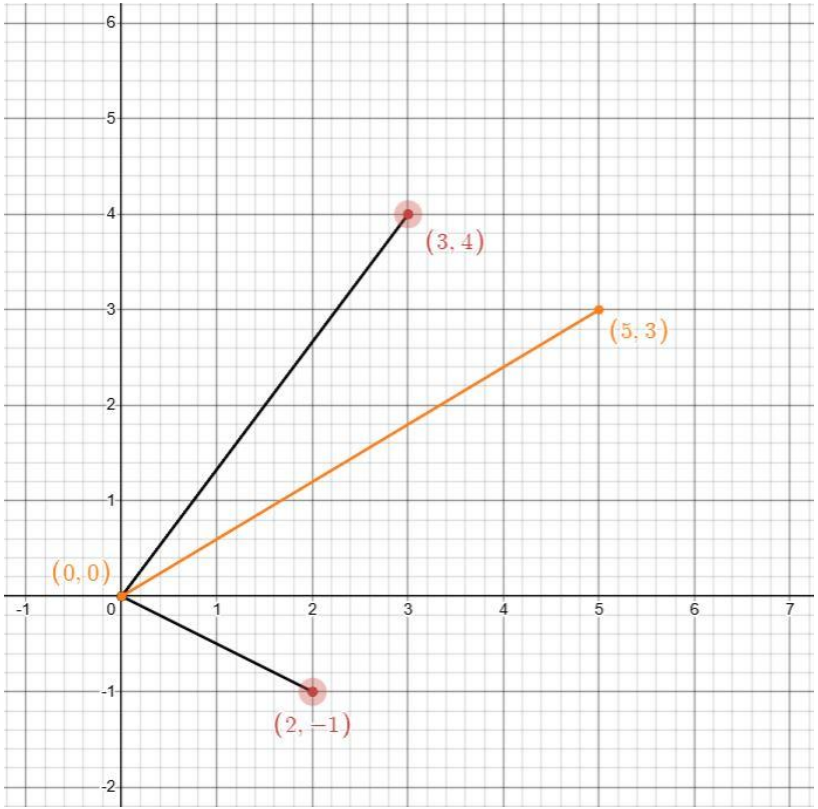
$c.x = 3 + 2 = 5$

$c.y = 4 + (-1) = 3$

Sehingga menghasilkan Vector baru,

yaitu vector

c = (5, 3).



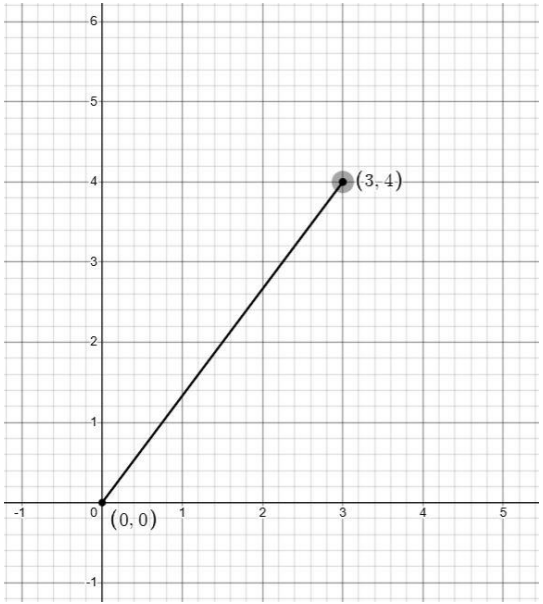
Operasi Vector | Perkalian Vector (Vector Multiplication) dengan Scalar Part 1

Perkalian Vector dengan *scalar* biasanya bertujuan untuk memperbesar atau memperkecil *magnitude* dan membalik (*flipping*) suatu Vector

Contoh mengalikan Vector dengan positif scalar 2:

a = [3, 4]

$scalar$ = 2



Operasi Vector | Perkalian Vector (Vector Multiplication) dengan Scalar Part 2

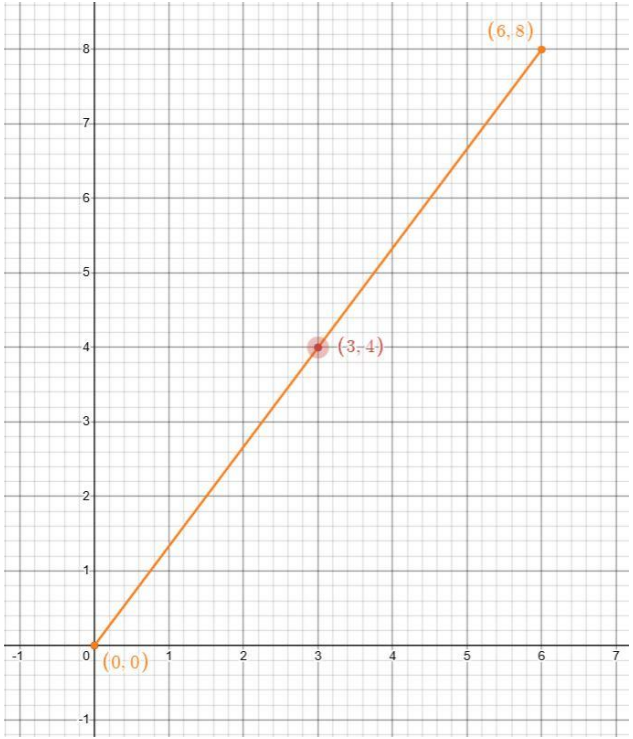
Mengalikan Vector a dengan ***scalar* 2** cukup mudah. Kalikan setiap elemennya dengan ***scalar* 2**.

$b.x$ = 3 * 2

$b.y$ = 4 * 2

Sehingga menghasilkan Vector baru, yaitu vector b.

Vector b = (6, 8).



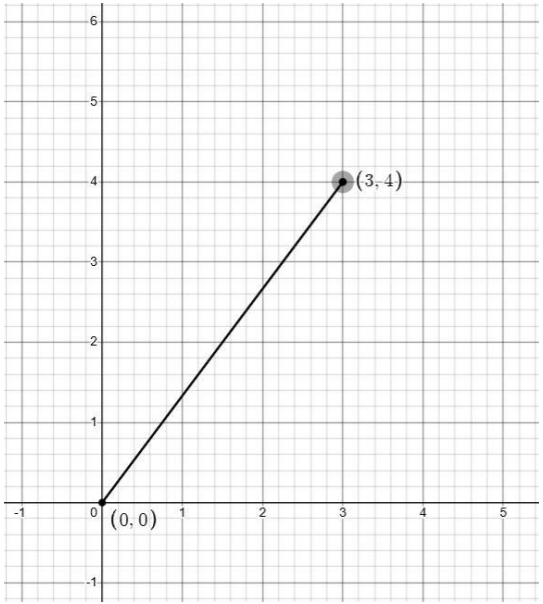
Operasi Vector | Perkalian Vector (Vector Multiplication) dengan Scalar Part 3

Perkalian Vector dengan ***negative scalar***. Mengalikan Vector dengan ***negative scalar* −1** akan membalik direction sebuah Vector.

Contoh mengalikan Vector dengan ***negative scalar* -1**:

a = [3, 4]

$scalar$ = -1



Operasi Vector | Perkalian Vector (Vector Multiplication) dengan Scalar Part 4

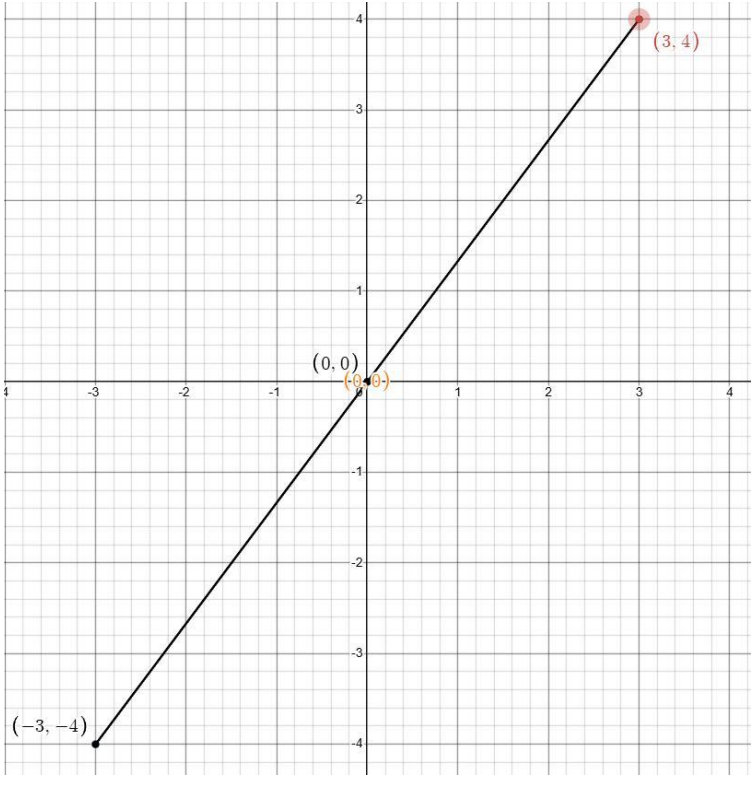
Mengalikan Vector a dengan ***negative scalar* -1** cukup mudah. Kalikan setiap elemennya dengan ***scalar* -1**.

$b.x$ = 3 * (-1)

$b.y$ = 4 * (-1)

Sehingga menghasilkan Vector baru, yaitu Vector b.

Vector b = (-3, -4).



Dot Product

Operasi Vector | Perkalian Vector dengan Vector (Dot Product) Part 1

Perkalian Vector dengan Vector (Dot Product) menghasilkan sebuah *scalar*, sehingga disebut juga *scalar product*. **Dot Product digunakan untuk mengukur kemiripan dua Vector**. Untuk menghitung Dot Product, ada dua cara yang bisa dilakukan.

Cara pertama kita bisa mengalikan magnitude dua Vector tersebut dengan nilai *cos(θ),* dimana θ adalah sudut antara dua Vector.

$$A.B = \|A\| \, \|B\| \, cos(\theta)$$

Cara kedua kita bisa menjumlahkan product dari setiap elemennya

$$A.B = A.x \times B.x + A.y \times B.y$$

Bentuk umum untuk N Dimensions Vector.

A = [a1, a2, a3, …, an]

B = [b1, b2, b3, …, bn]

$$A.B = \sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2} cos(\theta)$$

$$A.B = \sqrt{\sum_{i=1}^{n} A_i B_i}$$

Operasi Vector | Perkalian Vector dengan Vector (Dot Product) Part 2

Contoh **Dot Product**:

A = [3, 3]

B = [4, 1]

A.B = 3 * 4 + 3 * 1

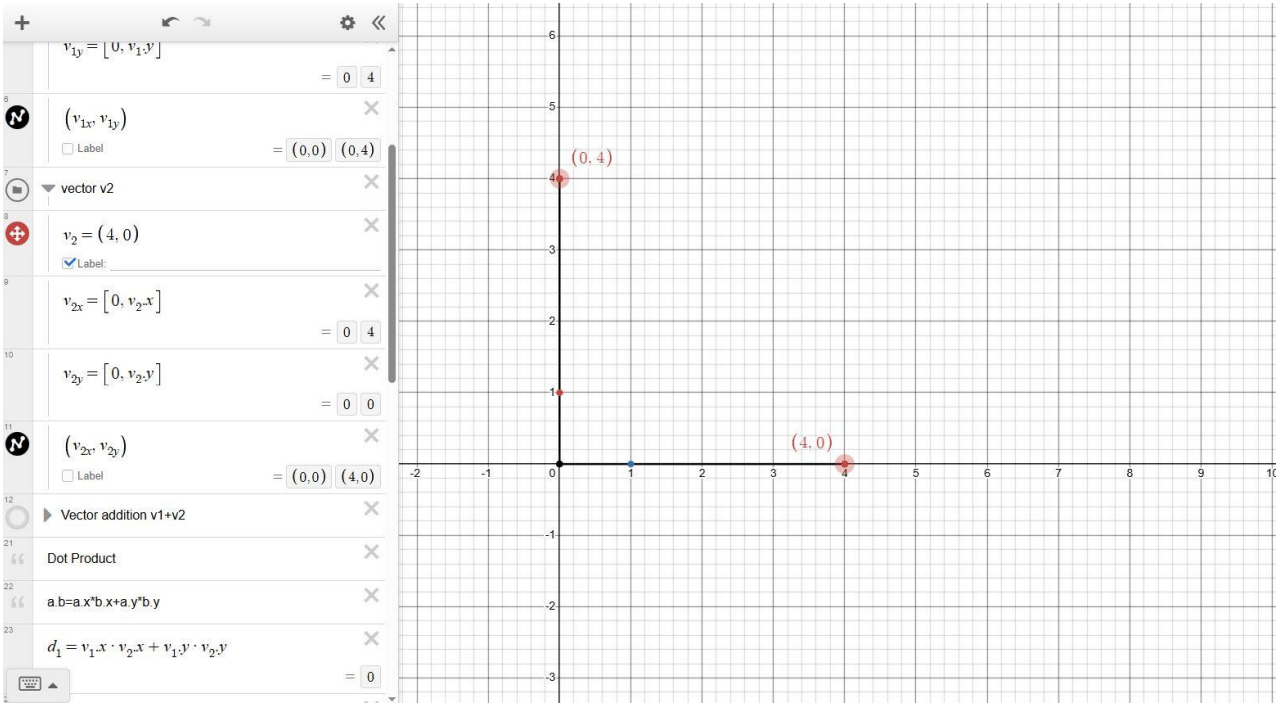
A.B = 15

Dot Product dari Vector A dan Vector B = 15

Operasi Vector | Perkalian Vector dengan Vector (Dot Product) Part 3

Properti dari Dot Product

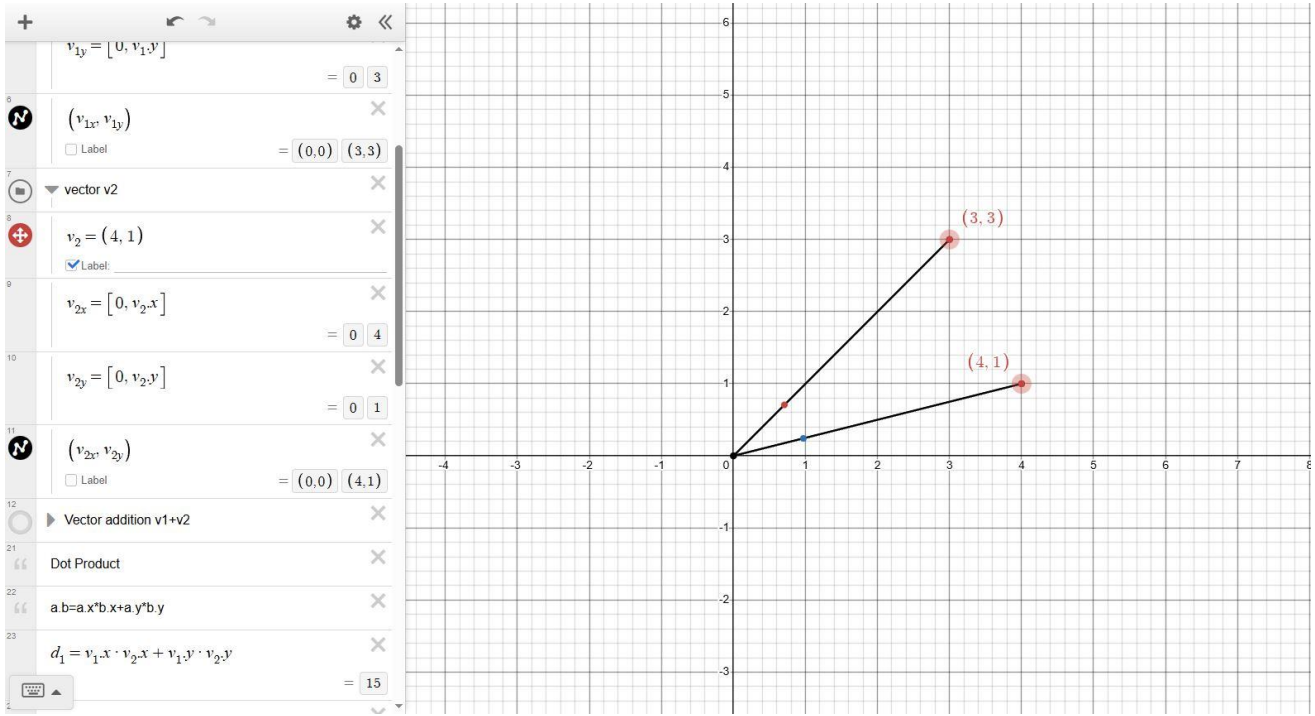
Dot Product bernilai 0 ketika sudut antara 2 Vector = 90°



Operasi Vector | Perkalian Vector dengan Vector (Dot Product) Part 4

Properti dari Dot Product

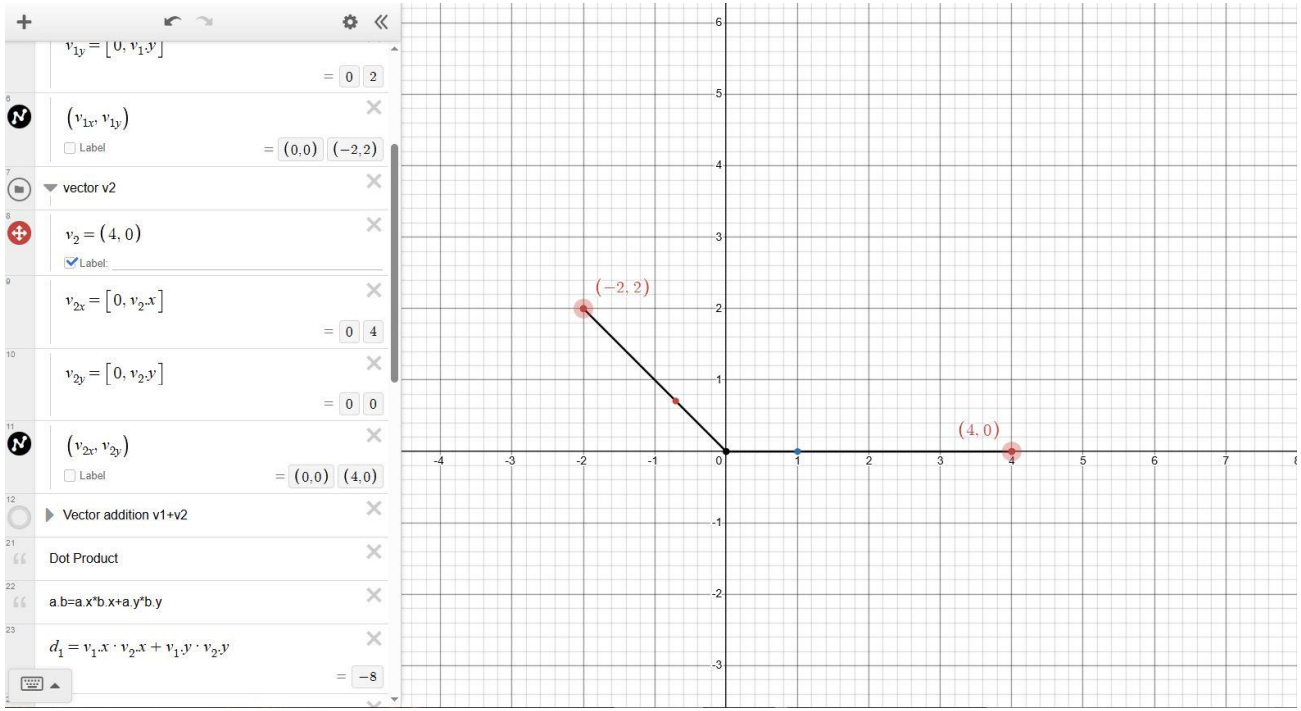
Dot Product bernilai lebih dari > 0 (*positive*) ketika sudut antara 2 Vector < (kurang dari) 90°



Operasi Vector | Perkalian Vector dengan Vector (Dot Product) Part 5

Properti dari Dot Product

Dot Product bernilai kurang dari < 0 (*negative*) ketika sudut antara 2 Vector > (lebih dari) 90°



Cross Product

Operasi Vector | Perkalian Vector Silang (Cross Product) Part 1

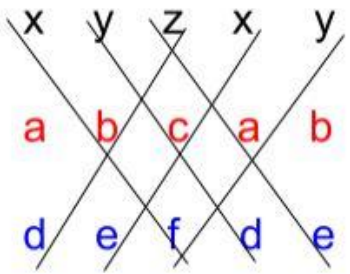
Perkalian Vector silang menghasilkan **Vector baru**, tidak seperti Dot Product, yang menghasilkan sebuah *scalar*. **Operasi Cross Product hanya bisa dilakukan pada sistem koordinat 3 Dimensi, sehingga hampir tidak pernah dimanfaatkan dalam dunia Machine Learning dan AI.** Cross Product biasanya banyak digunakan pada Game Engine dan Robotic.

Cara pertama kita bisa mengalikan magnitude dua Vector tersebut dengan nilai *sin(θ),* dimana θ adalah sudut antara dua Vector.

$$A \times B = \|A\| \, \|B\| \, sin(\theta)$$

Cara kedua kita bisa mengalikan silang setiap elemen pada dua Vector, kemudian melakukan operasi pengurangan dari setiap hasil perkalian silang tersebut.

A = [a, b, c] B = [d, e, f]



A x B = [(bf - ce), (cd - af), (ae - bd)]

Operasi Vector | Perkalian Vector Silang (Cross Product) Part 2

Contoh **Cross Product**:

A = [1, 2, 3]

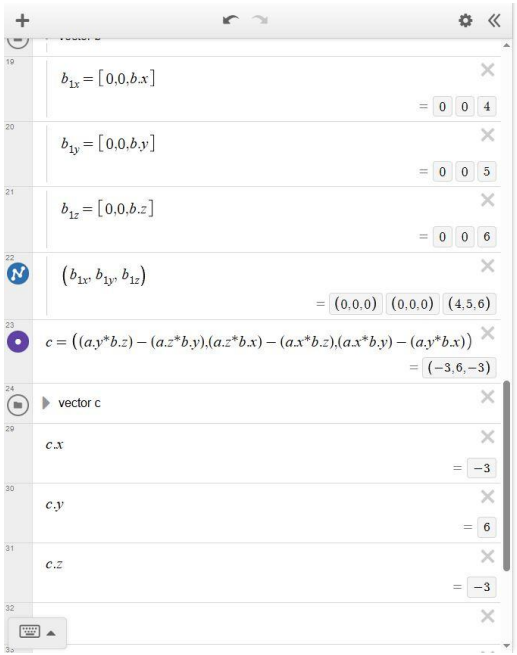
B = [4, 5, 6]

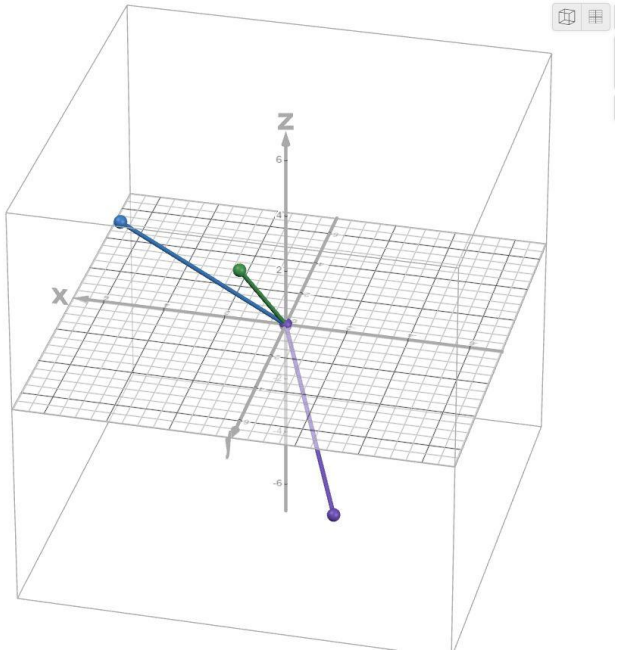
A x B = [(2 * 6) − (3 * 5), (3 * 4) − (1 * 6), (1 * 5) − (2 * 4)]

A x B = [12 − 15, 12 − 6, 5 − 8]

Cross Product dari Vector A dan Vector B menghasilkan Vector baru

= [− 3, 6, − 3]





Vector Metrics dan Vector Similarity

Vector Similarity

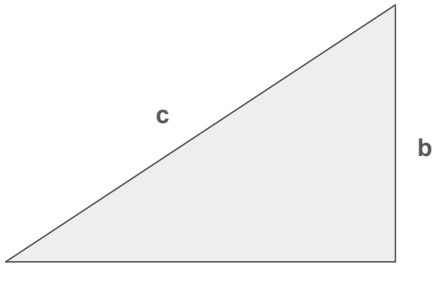
Vector Similarity adalah proses mengukur seberapa mirip dua koordinat data yang direpresentasikan menggunakan Vector dalam ruang multidimensi. Untuk mengukur seberapa mirip dua Vector kita menggunakan apa yang disebut sebagai **Vector Metrics**. **Vector Metrics** yang paling umum digunakan adalah **Euclidean Distance(L2 Distance)**, **Cosine Similarity, Cosine Distance**, dan **Dot Product/Inner Product (sudah kita bahas pada section sebelumnya)**.

Euclidean Distance (L2 Distance) Part 1

Euclidean Distance adalah Vector Metrics *based on* Teorema Pythagoras (Pythagorean Theorem). Mengukur kesamaan 2 Vector dengan Euclidean Distance itu seperti mengukur jalur terpendek dan terlurus dari 2 buah titik.

Jika merujuk kembali ke persamaan Teorema Pythagoras, persamaan Euclidean Distance tidak jauh berbeda. Persamaan ini hanya bentuk ekstensi dari persamaan Teorema Pythagoras untuk N Dimensions.

Persamaan Teorema Pythagoras (Pythagorean Theorem)



$$c^2 = a^2 + b^2$$

$$c = \sqrt{a^2 + b^2}$$

Euclidean Distance untuk N Dimensions.

$$E(p, q) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Dimana p dan q adalah 2 Koordinat Vector

Euclidean Distance (L2 Distance) Part 2

Untuk memahami kenapa **p (Koordinat Vector 1)** dikurangi terlebih dahulu dengan **q (Koordinat Vector 2)** sebelum melakukan perhitungan lebih lanjut pada persamaan Euclidean Distance, kita akan **simulasikan** pada **Vector *space* 2 Dimensions**.

$$E(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Jika kita memiliki 2 Vector, p dan q:

p = [6, 4]

q = [5, 1]

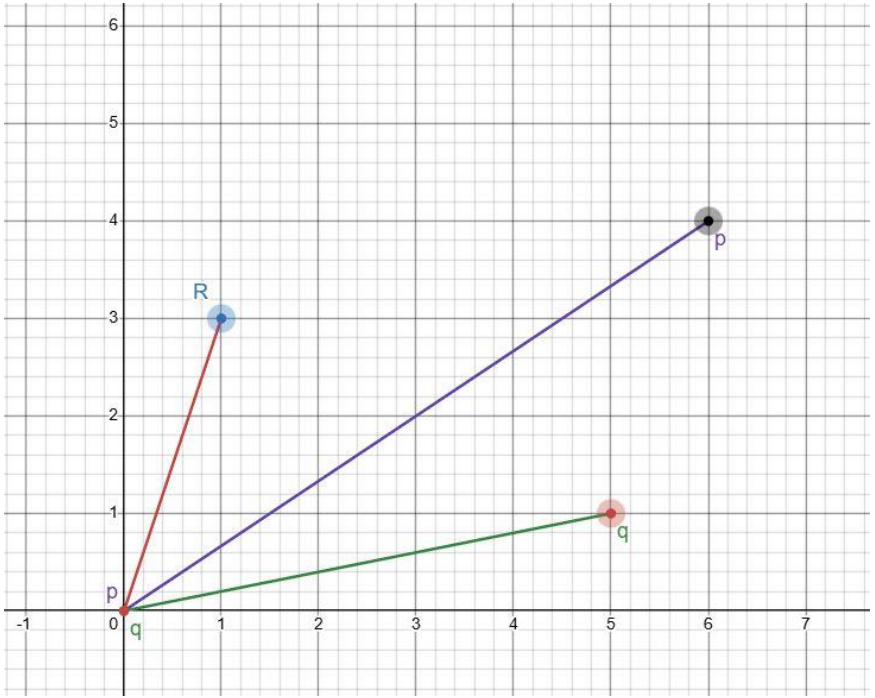
Kita akan melakukan operasi pengurangan pada Vector p dengan q

R = [(p1 - q1), (p2 - q2)]

R = [(6 - 5), (4 - 1)]

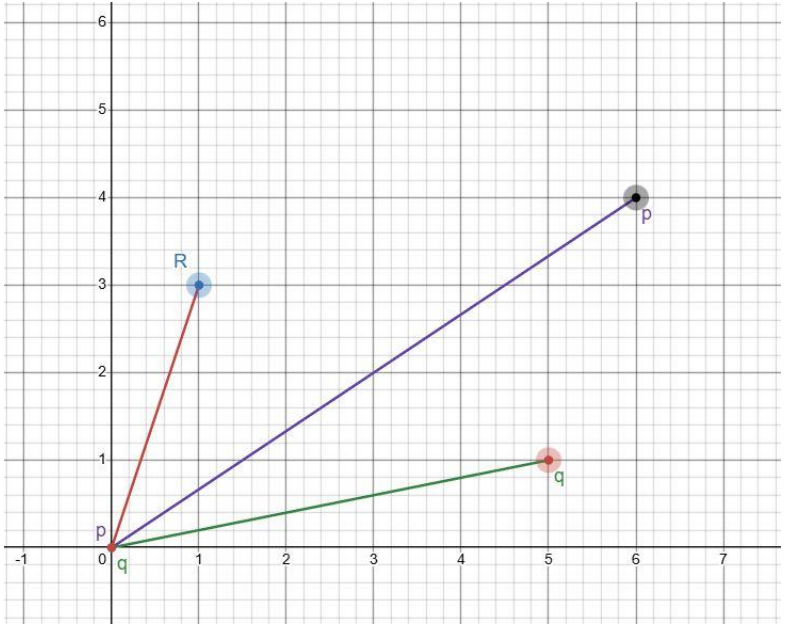
R = [1, 3]

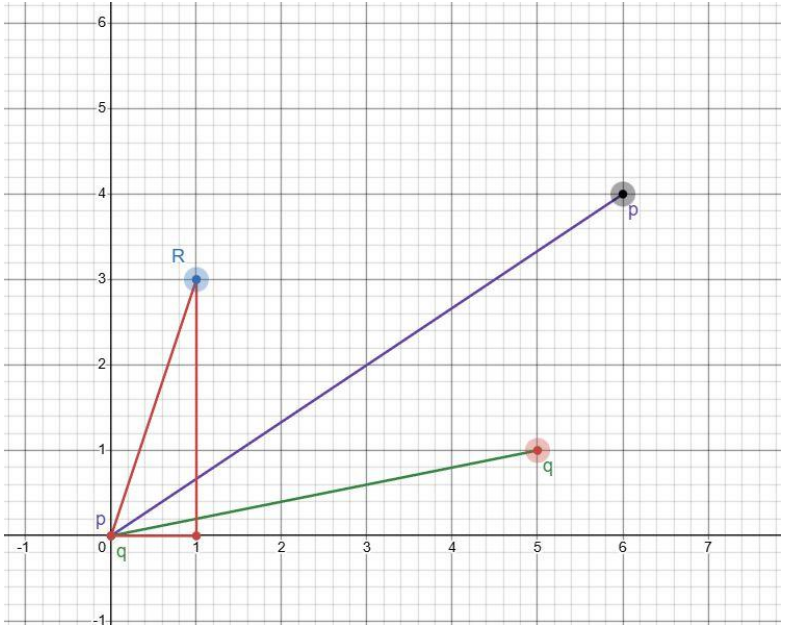
Hasil **R** (garis merah) adalah hasil ploting dari koordinat **R**, x = 1, y = 3.



Euclidean Distance (L2 Distance) Part 3

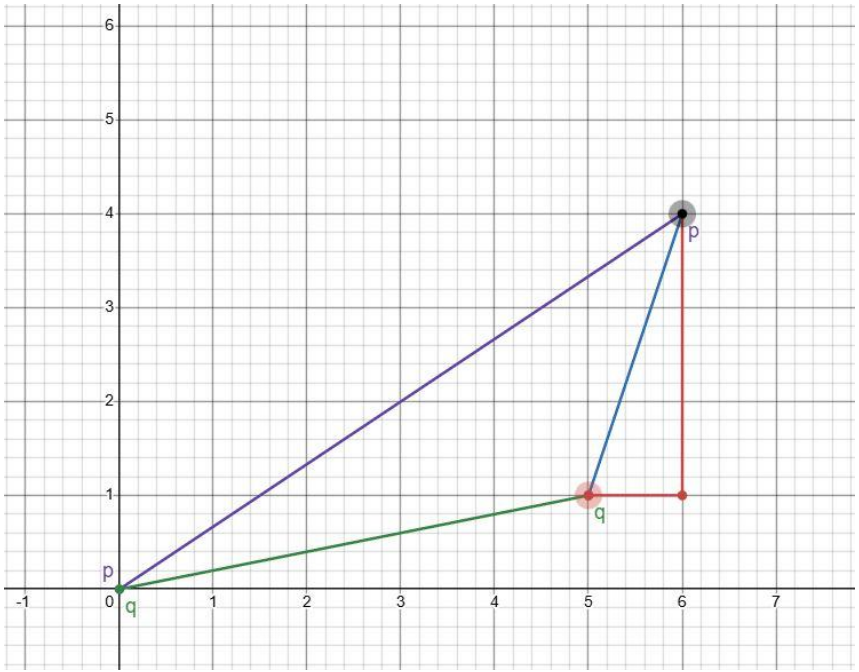
Koordinat **R** jika kita gambarkan penuh akan menjadi segitiga siku-siku (right angled triangle), **pada gambar sebelah kanan**.





Euclidean Distance (L2 Distance) Part 4

Koordinat **R** jika kita gambarkan penuh dan kita **tempatkan pada posisi koordinat q dan p sebagai origin**, maka akan **mensimulasikan dengan lebih jelas lagi garis mana sebenarnya yang akan kita ukur untuk Vector Metrics Euclidean Distance**. Kemudian **garis berwarna biru** itulah yang akan kita tetapkan sebagai **Euclidean Distance dari Vector p dan Vector q**.



$$E(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Menggunakan contoh sebelumnya, jika kita memiliki 2 Vector, p dan q:

p = [6, 4]

q = [5, 1]

$$E(p,q) = \sqrt{(6-5)^2 + (4-1)^2}$$

$$E(p,q) = 3.16$$

Sehingga distance dari **p** ke **q** = ~3.16

Euclidean Distance (L2 Distance) Part 5

Contoh mengukur kemiripan Vector dengan beberapa Vector lain menggunakan Euclidean Distance.

Katakanlah kita mempunyai 2 koordinat pada dataset yang kita miliki.

b = [12, 11]

c = [10, 0. 5]

Kemudian kita mempunyai data koordinat baru,

Yaitu:

a = [7, 3]

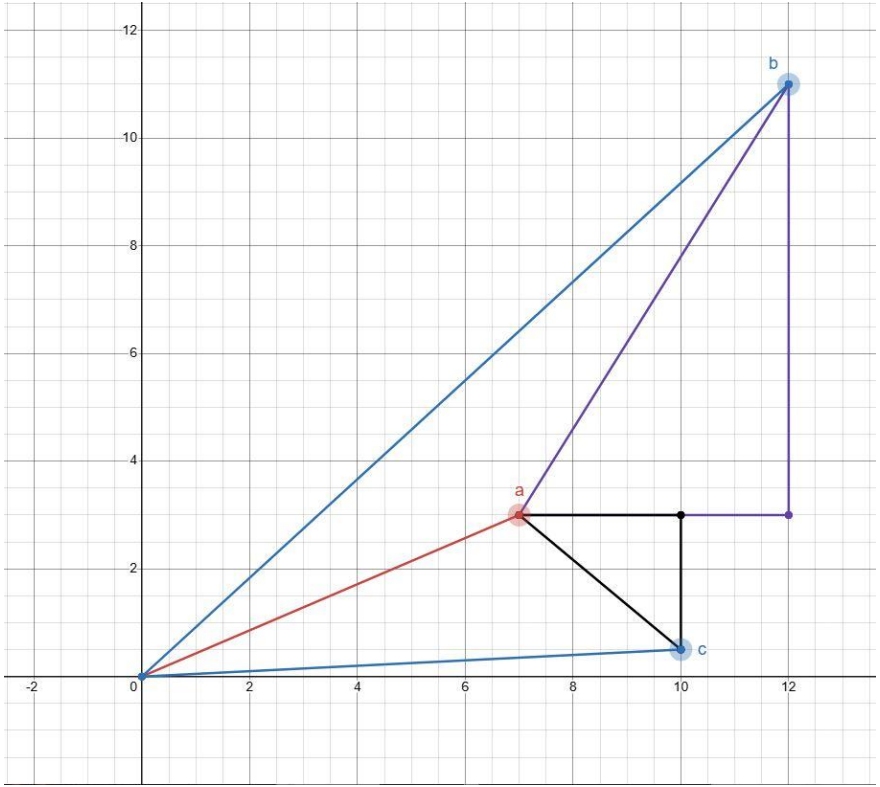
Kita akan ukur seberapa dekat data-data yang ada di dataset dengan data baru **a** menggunakan **Euclidean Distance**.

Jarak dari a ke b.

$$dab = \sqrt{(12-7)^2 + (11-3)^2}$$
$$dab = 9.43$$

Jarak dari a ke c

$$dac = \sqrt{(10-7)^2 + (0.5-3)^2}$$
$$dac = 3.90$$



Jarak Vector a ke b = 9.43 dan Vector a ke c = 3.90. Sehingga bisa disimpulkan, jarak a ke c **lebih dekat** dari jarak a ke b, seperti yang terlihat pada gambar diatas.

Cosine Distance dan Cosine Similarity Part 1

Cosine Distance dan Cosine Similarity adalah Vector Metrics yang mengukur kesamaan 2 Vector berdasarkan **sudut (*angle*) function Cos (Cosine)** antara 2 Vector. Berbeda dengan **Euclidean Distance atau Dot Product** yang bergantung pada besaran/panjang (*magnitude*) untuk menyatakan 2 Vector memiliki kemiripan. **Cosine Distance dan Cosine Similarity** menggunakan sudut (*angle*) untuk menyatakan 2 Vector memiliki kemiripan. Sehingga pada **Cosine Distance dan Cosine Similarity, arah (*direction*) lebih penting dari pada besaran/panjang(*magnitude*)**.

Cosine Distance dan Cosine Similarity Part 2

Pada persamaan **Cosine Similarity** dibawah ini akan menghasilkan nilai **-1 sampai 1**. **Cosine Similarity menyatakan bahwa semakin besar nilai cos(θ), semakin mirip 2 Vector tersebut.**

Jika Nilai cos(θ) = 1, menandakan 2 Vector searah, mirip dari sisi arah (direction).

Jika Nilai cos(θ) = 0, (sudut 90°, Vector tegak lurus/*perpendicular*), menandakan 2 Vector tidak mirip.

Jika Nilai cos(θ) = -1, (sudut 180°, Vector berlawanan arah), menandakan 2 Vector sangat tidak mirip.

Persamaan Cosine Similarity	Cosine Similarity untuk N Dimensions.

$$cos(\theta) = \frac{A.B}{\|A\| \, \|B\|}$$

$$cos(\theta) = \frac{\sum_{i=1}^{n} AiBi}{\sqrt{\sum_{i=1}^{n} Ai^2}\sqrt{\sum_{i=1}^{n} Bi^2}}$$

Dimana

A.B = Dot Product Vector A dan Vector B

||A|| = Magnitude Vector A

||B|| = Magnitude Vector B

Cosine Distance dan Cosine Similarity Part 3

Untuk memahami kenapa **perhitungan Cosine Similarity** kita akan **simulasikan** pada **Vector *space* 2 Dimensions**.

$$cos(\theta) = \frac{\sum_{i=1}^{n} AiBi}{\sqrt{\sum_{i=1}^{n} Ai^2}\sqrt{\sum_{i=1}^{n} Bi^2}}$$

Jika kita memiliki 2 Vector, p dan q:

a = [3, 3]

b = [6, 2]

$$cos(\theta) = \frac{3 \times 6 + 3 \times 2}{\sqrt{3^2 + 3^2} \times \sqrt{6^2 + 2^2}}$$

$$cos(\theta) = 0.894427190999916$$

Sehingga Cosine Similarity dari **a** ke **b** = ~0.89

Cosine Distance dan Cosine Similarity Part 4

Contoh mengukur kemiripan Vector dengan beberapa Vector lain menggunakan Cosine Similarity.

Katakanlah kita mempunyai 2 koordinat pada dataset yang kita miliki.

b = [6, 2]

c = [7, -4]

Kemudian kita mempunyai data koordinat baru,

Yaitu:

a = [3, 3]

Kita akan ukur seberapa dekat data-data yang ada di dataset dengan data baru **a** menggunakan **Cosine Similarity**.

cos(θ) **a** dengan **b**

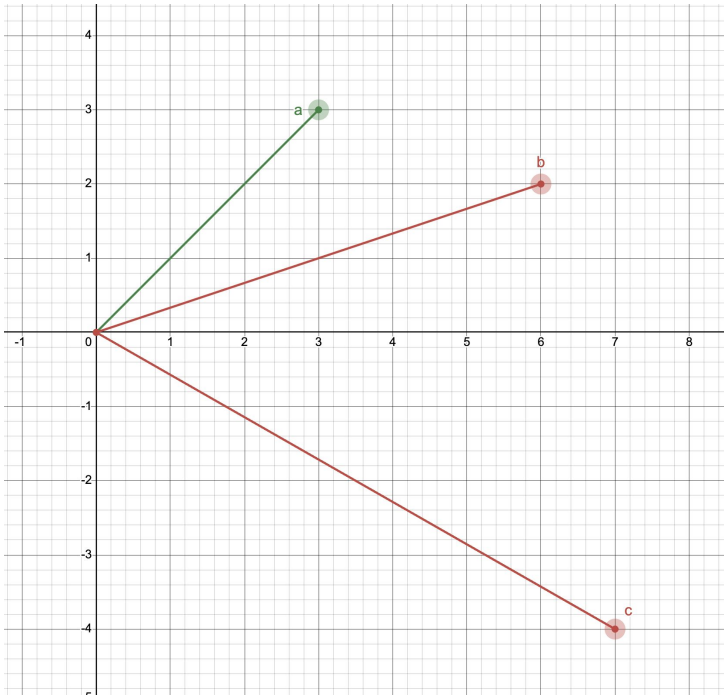
$$cos(\theta) = \frac{3 \times 6 + 3 \times 2}{\sqrt{3^2 + 3^2} \times \sqrt{6^2 + 2^2}}$$

$$cos(\theta) = 0.894427190999916$$

cos(θ) **a** dengan **c**

$$cos(\theta) = \frac{3 \times 7 + 3 \times (-4)}{\sqrt{3^2 + 3^2} \times \sqrt{7^2 + (-4)^2}}$$

$$cos(\theta) = 0.2631174057921088$$



Cosine Similarity Vector **a ke b = ~0.89** dan Vector **a ke c = ~0.26**. Sehingga bisa disimpulkan, Vector **a lebih mirip dengan b (nilai cos(θ) a dengan b lebih besar)** dibandingkan dengan Vector **a** dengan **c (nilai cos(θ) a dengan c lebih kecil)**.

Cosine Distance dan Cosine Similarity Part 5

Pada persamaan **Cosine Distance** dibawah ini akan menghasilkan nilai **0 sampai 2, atau 0 sampai 1 (jika Vector sudah di normalisasi. Baca bagian Unit Vector !**). **Cosine Distance adalah ekstensi dari Cosine Similarity untuk menyatakan bahwa semakin kecil nilai 1-cos(θ), semakin mirip 2 Vector tersebut.**

Jika Nilai 1-cos(θ) = 0, menandakan 2 Vector searah, mirip dari sisi arah (direction).

Jika Nilai 1-cos(θ) = 1, (sudut 90°, Vector tegak lurus/*perpendicular*), menandakan 2 Vector tidak mirip.

Jika Nilai 1-cos(θ) = 2 (1- (-1) = 2), (sudut 180°, Vector berlawanan arah), menandakan 2 Vector sangat tidak mirip.

Persamaan Cosine Distance

$$D_{cos} = 1 - \frac{A.B}{\|A\| \, \|B\|}$$

Cosine Distance untuk N Dimensions.

$$D_{cos} = 1 - \frac{\sum_{i=1}^{n} AiBi}{\sqrt{\sum_{i=1}^{n} Ai^2}\sqrt{\sum_{i=1}^{n} Bi^2}}$$

Dimana

A.B = Dot Product Vector A dan Vector B

||A|| = Magnitude Vector A

||B|| = Magnitude Vector B

Cosine Distance dan Cosine Similarity Part 6

Contoh mengukur kemiripan Vector dengan beberapa Vector lain menggunakan Cosine Distance.

Katakanlah kita mempunyai 2 koordinat pada dataset yang kita miliki.

b = [6, 2]

c = [7, -4]

Kemudian kita mempunyai data koordinat baru,

Yaitu:

a = [3, 3]

Kita akan ukur seberapa dekat data-data yang ada di dataset dengan data baru **a** menggunakan **Cosine Distance**.

Dcos **a** dengan **b**

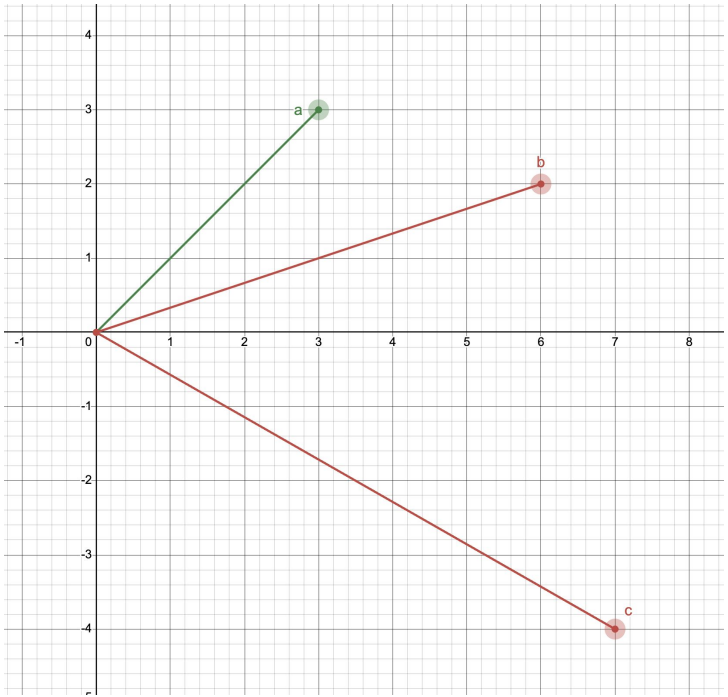
$$D_{cos} = 1 - \frac{3 \times 6 + 3 \times 2}{\sqrt{3^2 + 3^2} \times \sqrt{6^2 + 2^2}}$$

$$D_{cos} = 0.10557280900008403$$

Dcos **a** dengan **c**

$$D_{cos} = 1 - \frac{3 \times 7 + 3 \times (-4)}{\sqrt{3^2 + 3^2} \times \sqrt{7^2 + (-4)^2}}$$

$$D_{cos} = 0.7368825942078912$$



Cosine Distance **a ke b = ~0.10** dan **a ke c = ~0.73**. Sehingga bisa disimpulkan, Vector **a lebih mirip dengan b (nilai Dcos a dengan b lebih kecil)** dibandingkan dengan Vector **a** dengan **c (nilai Dcos a dengan c lebih besar)**.

Cosine Distance dan Cosine Similarity Part 7

Tabel perbandingan Cosine Similarity dan Cosine Distance

	Cosine Similarity	Cosine Distance
Makna nilai	Semakin besar semakin mirip	Semakin kecil semakin mirip
Tujuan	Mengukur kemiripan	Mengukur perbedaan/jarak
Kegunaan	Analisa kemiripan “makna” Text	Algoritma Clustering: KNN, K-Means, Approximate Nearest Neighbors seperti HNSW (Hierarchical Navigable Small World), IVF (Inverted File Index).

Cosine Distance dan Cosine Similarity Part 8

Mengapa fungsi *Trigonometry* cos(θ) muncul pada Cosine Similarity dan Dot Product ?

Pada pembahasan Dot Product pada bagian sebelumnya, persamaan di bawah ini sebenarnya memberitahu kita seberapa jauh “bayangan” Vector B yang bekerja (“jatuh”) searah dengan Vector A

$$A.B = ||A||\,||B||\,cos(\theta)$$

Pembahasan **Trigonometry** pada bagian sebelumnya, menyatakan bahwa:

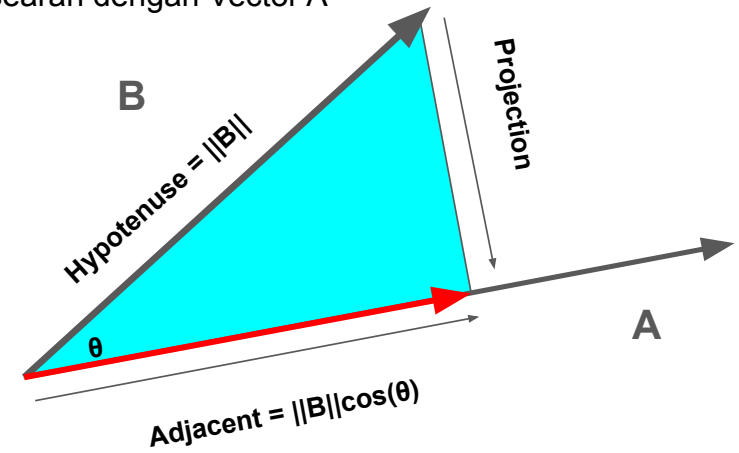
$$cos(\theta) = \frac{Adjacent}{Hypotenuse}$$

$$Adjacent = Hypotenuse \times cos(\theta)$$

Dimana:
Hypotenuse = Magnitude dari Vector B, ||B||
cos(θ) = nilai **cos** sudut (angle) antara Vector A dan Vector B

Sehingga proyeksi/Adjacent untuk komponen Vector B bisa dihitung dengan:
Adjacent = ||B||cos(θ)

Oleh karena itu, persamaan $A.B = ||A||\,||B||\,cos(\theta)$ secara logis mengalikan panjang(magnitude) **Vector A** dengan **komponen Vector B** (hasil proyeksi/Adjacent/**panah (arrow) berwarna merah**) yang sudah "diluruskan" ke arah Vector A.



Hasil proyeksi/Adjacent (panah (arrow) berwarna merah) bisa dianalogikan sebagai hasil proyeksi (projection) atau bayangan Vector B terhadap Vector A.

Cosine Distance dan Cosine Similarity Part 9

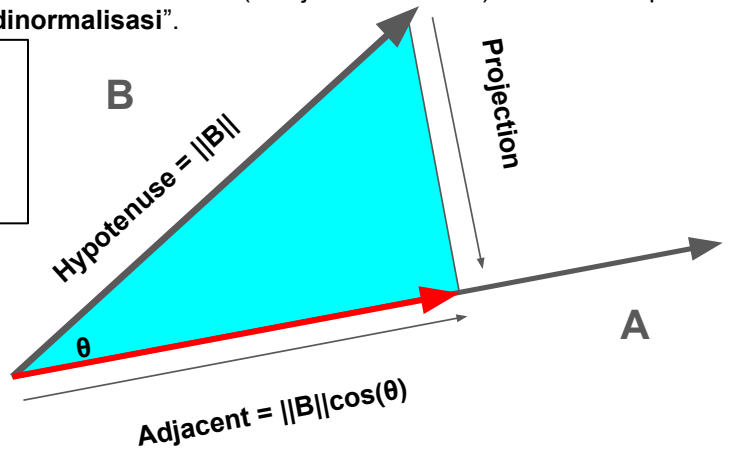
Mengapa fungsi *Trigonometry* cos(θ) muncul pada Cosine Similarity dan Dot Product ?

Pada pembahasan **Cosine Similarity** pada bagian sebelumnya, kemudian merujuk pada persamaan **Trigonometry untuk mencari cos(θ)**, persamaan **Cosine Similarity** dibawah ini sebenarnya adalah bentuk perubahan posisi komponen dari persamaan **Dot Product**. Bahkan operasi Dot Product-nya akan menghasilkan hasil yang sama jika Vector telah dinormalisasi (menjadi Unit Vector). Lebih detail pada bagian "**Cosine Similarity adalah Dot Product dari dua Vector yang sudah dinormalisasi**".

Pembahasan **Trigonometry** pada bagian sebelumnya, menyatakan bahwa:

$$cos(\theta) = \frac{Adjacent}{Hypotenuse}$$

$$Adjacent = Hypotenuse \times cos(\theta)$$



Cosine Similarity

$$cos(\theta) = \frac{A.B}{\|A\| \, \|B\|}$$

Dot Product

$$A.B = \|A\| \, \|B\| \, cos(\theta)$$

Dimana

A.B = Dot Product Vector A dan Vector B

||A|| = Magnitude Vector A

||B|| = Magnitude Vector B

||A|| ||B|| = Total magnitude Vector A dan Vector B

cos(θ) = nilai **cos** sudut (angle) antara Vector A dan Vector B

Cosine Distance dan Cosine Similarity Part 10

Cosine Similarity adalah Dot Product dari dua Vector yang sudah dinormalisasi

Untuk membuktikan pernyataan: **Cosine Similarity adalah Dot Product dari dua Vector yang sudah dinormalisasi**, kita akan menghitung **Cosine Similarity** dan **Dot Product (dengan setiap Vector-nya dinormalisasi/ menjadi Unit Vector)**.

Menghitung **Cosine Similarity** Vector A dan B:

A = [6, 2]

B = [3, 3]

cos(θ) **A** dengan **B**

$$cos(\theta) = \frac{\sum_{i=1}^{n} AiBi}{\sqrt{\sum_{i=1}^{n} Ai^2}\sqrt{\sum_{i=1}^{n} Bi^2}}$$

$$cos(\theta) = \frac{6 \times 3 + 2 \times 3}{\sqrt{6^2 + 2^2} \times \sqrt{3^2 + 3^2}}$$

$$cos(\theta) = 0.894427190999916$$

=

=

Menghitung **Dot Product** Vector A dan B yang sudah **dinormalisasi** menjadi **Unit Vector**:

A = [6, 2]

B = [3, 3]

Dot Product **A** dengan **B dan sudah dinormalisasi dalam Unit Vector**

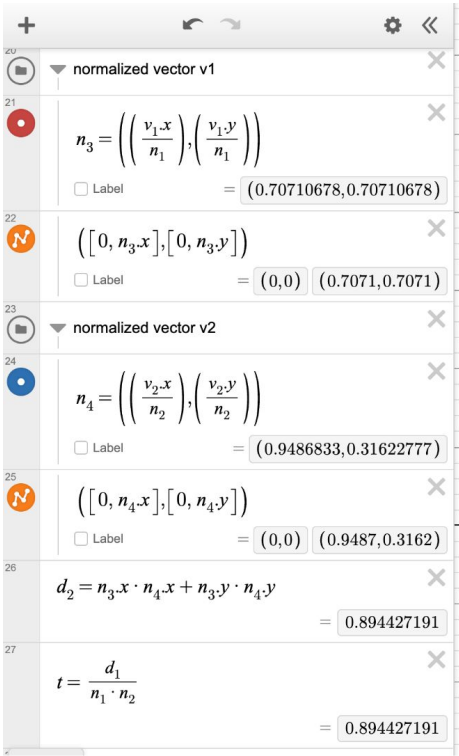
$$\hat{A}.\hat{B} = \left(\frac{A_x}{\sqrt{\sum_{i=1}^{n} A^2}}\right) \times \left(\frac{B_x}{\sqrt{\sum_{i=1}^{n} B^2}}\right) + \left(\frac{A_y}{\sqrt{\sum_{i=1}^{n} A^2}}\right) \times \left(\frac{B_y}{\sqrt{\sum_{i=1}^{n} B^2}}\right) ... + \left(\frac{A_n}{\sqrt{\sum_{i=1}^{n} A^2}}\right) \times \left(\frac{B_n}{\sqrt{\sum_{i=1}^{n} B^2}}\right)$$

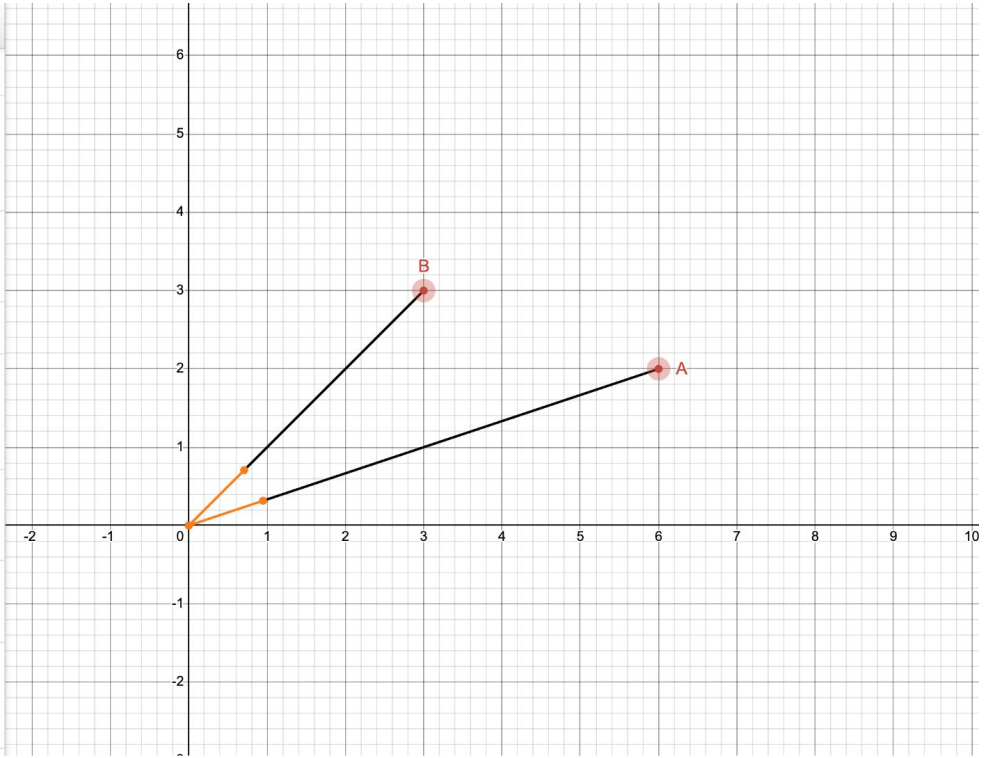
$$\hat{A}.\hat{B} = \frac{6}{\sqrt{6^2 + 2^2}} \times \frac{3}{\sqrt{3^2 + 3^2}} + \frac{2}{\sqrt{6^2 + 2^2}} \times \frac{3}{\sqrt{3^2 + 3^2}}$$

$$\hat{A}.\hat{B} = 0.894427190999916$$

Cosine Distance dan Cosine Similarity Part 11

Cosine Similarity adalah Dot Product dari dua Vector yang sudah dinormalisasi





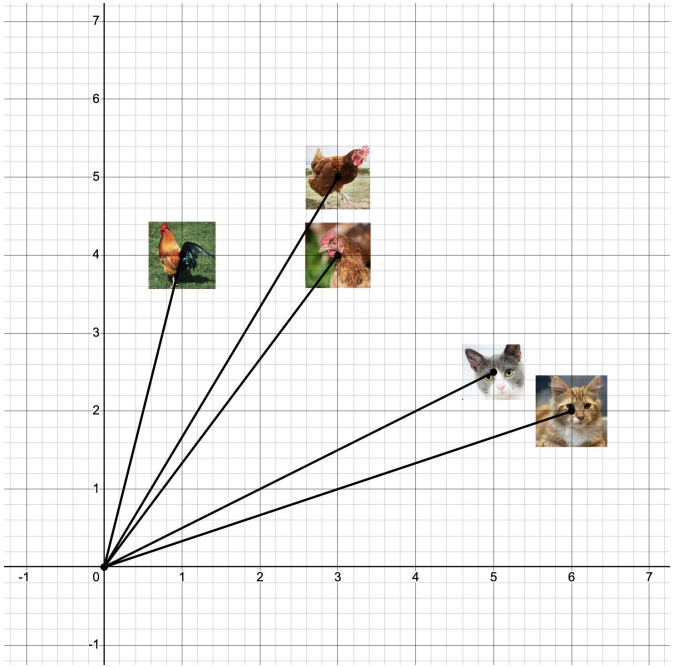
Vector warna orange, adalah Vector A dan Vector B yang sudah dinormalisasi (dalam Unit Vector)

Perhitungan pada bagian sebelumnya, kita coba plot pada graph. **Hasil perhitungan Cosine Similarity sama dengan hasil perhitungan Dot Product dengan setiap Vectornya sudah dinormalisasi (dalam Unit Vector)**.

Vector Embedding

Vector Embedding Part 1

Dalam konteks *Machine Learning* dan *Artificial Intelligence*, **Vector Embedding** adalah representasi numerik dari data non-numerik (seperti teks, gambar, atau audio) ke dalam ruang Vector berdimensi tinggi (*high-dimensional space*). Proses ini bertujuan untuk memetakan informasi *semantic* (makna) ke dalam koordinat Matematis. **Dua data yang memiliki kemiripan makna akan ditempatkan pada posisi yang berdekatan dalam ruang Vector tersebut**. **Vector Embedding tidak ditentukan secara manual oleh manusia**, melainkan dihasilkan melalui proses **Feature Extraction** oleh model ***Deep Learning***.



Data	Representasi Vector
	[5, 2.5]
	[6, 2]
	[1, 4]
	[3, 5]
	[3, 4]

Vector Embedding Part 2 | Evolusi Representasi Data (History & Development)

Era Computer Vision (AlexNet - 2012 | Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton)

Sejarah modern Vector Embedding dimulai secara masif pada tahun 2012 melalui paper *ImageNet Classification with Deep Convolutional Neural Networks* **atau yang lebih kita kenal sebagai AlexNet**. Berikut hasil observasi yang dilakukan oleh para penulis Paper tersebut.

- **Terobosan:** Model ini membuktikan bahwa *Convolutional Neural Networks* (CNN) dapat **mengekstraksi dan belajar fitur visual secara otomatis**.
- **Hasil:** Gambar mentah diubah menjadi *feature vector* (embedding) berdimensi 4096. Untuk pertama kalinya, **gambar dapat dibandingkan secara matematis melalui jarak antar Vector, bukan sekadar perbandingan *pixel* mentah**.

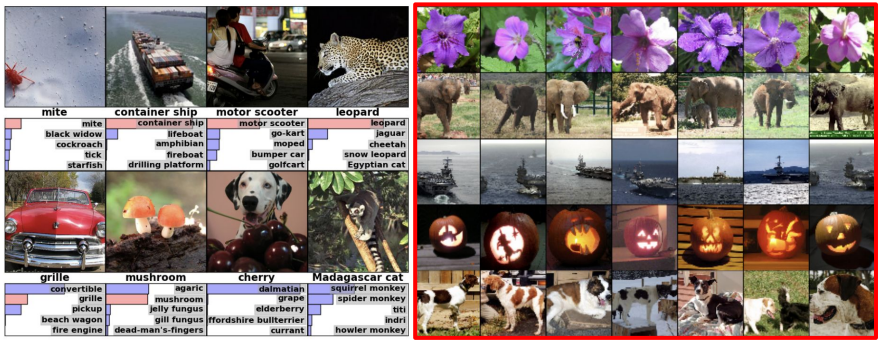


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf ImageNet Classification with Deep Convolutional Neural Networks | University of Toronto

Pada Section 6.1, dalam Paper tersebut, para penulis melakukan eksperimen lain, selain eksperimen utama dari Paper ini, yaitu Image Classification. Perhatikan *statement*:
"(Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image."

- **Eksperimen Image Retrieval:** Penulis menguji kemampuan model dengan mengambil 5 gambar dari *test set* (kolom pertama) dan mencari gambar dari *training set* yang memiliki **jarak Euclidean Distance terkecil** pada Vector fitur 4096-dimensi.
- **Keunggulan Ruang Vector (Embedding Space):** Hasil menunjukkan bahwa: meskipun gambar-gambar tersebut memiliki perbedaan pada level *pixel*, seperti pose, latar belakang, dan sudut pandang yang berbeda, model tetap mampu menemukan objek yang **secara semantik** sama (seperti gajah, anjing, atau kapal).
- Eksperimen ini membuktikan bahwa lapisan tersembunyi (*hidden layer*) terakhir model tidak sekadar melihat "warna" atau "*pixel*", melainkan membentuk **pemahaman konsep objek** di dalam ruang Vector.

Vector Embedding Part 3 | Evolusi Representasi Data (History & Development)

Era Natural Language Processing (Word2Vec - 2013 | Google)

Setelah gambar, dunia NLP mengalami revolusi melalui **Word2Vec**. Word2Vec memungkinkan kata direpresentasikan menggunakan Vector (Word Embedding).

Kekurangan dari model Word Embedding:

- Tidak *Context aware*. 1 kata (*word*) direpresentasikan dengan 1 Vector, sehingga saat proses analisis dengan model ini, model tidak melihat *surrounding word*.
- Representasi Vector selalu sama, sekalipun konteks berbeda. Misal kata **Bank** (**tempat menyimpan uang)** dengan **Bank (tepi sungai)** akan memiliki representasi Vector yang sama. Contoh lain kata dalam Bahasa Indonesia adalah **Bisa (mampu)** dengan **Bisa (Racun pada Ular)**.
- Model tidak bisa mewakili 1 kalimat secara alami. Biasanya butuh proses agregasi, sebab setiap kata pada kalimat akan memiliki Vector masing-masing.

Vector Embedding Part 4 | Evolusi Representasi Data (History & Development)

Era Transformers & Attention (Paper: Attention Is All You Need, model BERT & Sentence Transformers - 2017 | Google)

https://arxiv.org/abs/1706.03762 Paper ini memperkenalkan arsitektur **Transformer** yang menggantikan mekanisme sekuensial (seperti RNN) dengan mekanisme **Self-Attention**. Hal ini berdampak signifikan pada kualitas *Vector Embedding*:

- **Mekanisme Self-Attention:** Tidak seperti model sebelumnya yang memproses kata satu per satu, Transformer memungkinkan setiap elemen dalam input (kata atau *patch* gambar) untuk "memperhatikan" (*attend to*) elemen lainnya secara simultan.
- **Contextualized Embeddings:** Inilah perbedaan terbesarnya. Melalui *Self-Attention*, Vector yang dihasilkan bersifat dinamis (sesuai konteks).
    - *Contoh:* Kata **"Bank"** dalam kalimat *"I want to save money in the Bank"* dan *"I was standing near the river Bank"*, kata **"Bisa"** (**Bahasa Indonesia**) dalam kalimat ***"Saya Bisa melakukannya"*** dan *"Saya keracunan Bisa Ular"* akan menghasilkan koordinat Vector yang **berbeda** karena ***model* memperhatikan kata-kata di sekitarnya**.
- **Sentence Embeddings:** Seluruh kalimat atau paragraf kini dapat diringkas menjadi satu Vector tunggal yang merepresentasikan seluruh ide pikiran.

Vector Embedding Part 5 | Evolusi Representasi Data (History & Development)

Era Modern (adaptasi Arsitektur Transformers untuk Computer Vision): Vision Transformers (ViT) & Multi-Modal (CLIP)

ViT adalah adaptasi langsung dari arsitektur Transformer (dari paper ***Attention Is All You Need***) dengan judul paper ***An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*** (https://arxiv.org/abs/2010.11929) untuk data gambar.

- **Mekanisme "Patches":** Berbeda dengan CNN(*Convolutional Neural Networks*) yang memproses gambar dengan *sliding window*, ViT memecah gambar menjadi potongan-potongan kecil (*patches*) dan memperlakukannya seperti urutan kata dalam sebuah kalimat.
- **Global Context:** Melalui *Self-Attention*, setiap bagian gambar dapat berinteraksi dengan bagian lainnya secara global. Hal ini menghasilkan Embedding yang sangat baik dibandingkan CNN yang bersifat lokal.

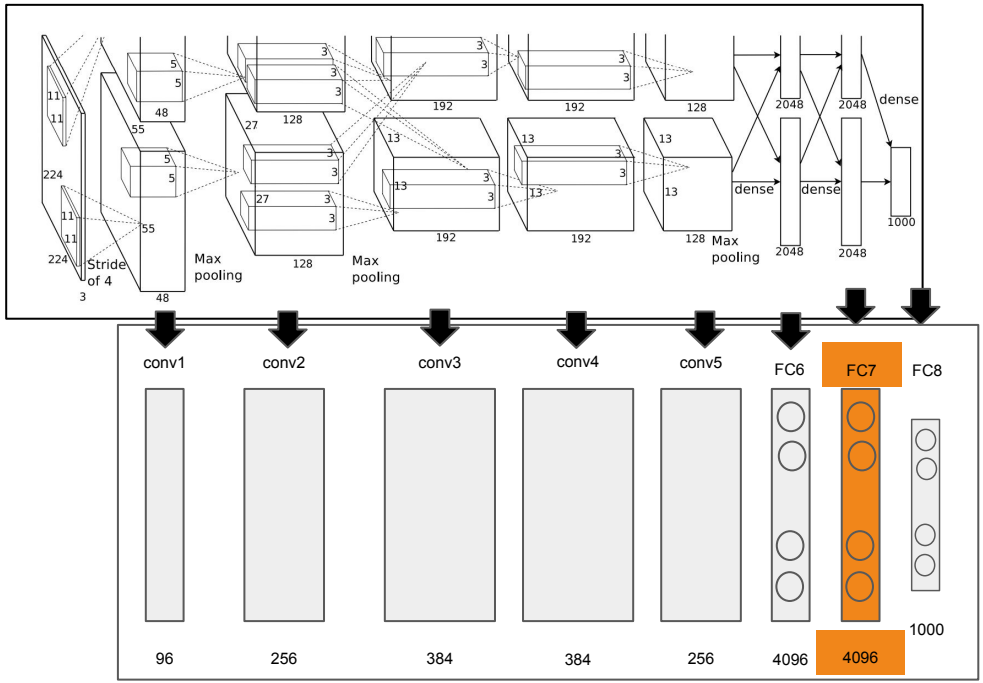
Selain itu, perkembangan terkini telah mencapai tahap **Multi-Modal Learning**, di mana model seperti **CLIP** (*Contrastive Language-Image Pre-training*) dari **OpenAI** mampu menyatukan dua dunia yang berbeda. CLIP memetakan **teks** dan **gambar** ke dalam **ruang Vector yang sama (*Shared Vector Space*)**. Hal ini memungkinkan sistem untuk **mencari gambar menggunakan deskripsi teks (dan sebaliknya) hanya dengan menghitung *Cosine Similarity* antara Vector teks dan Vector gambar**.

Vector Embedding Part 6 | Dimana Vector Embedding diambil dari Model

Secara umum Deep Learning Model memiliki 2 peran,

- Memahami data (merepresentasikan data)
- Mengambil/Membuat keputusan (*decision*)

Untuk memahami secara visual **dimana Vector Embedding diambil**, kita akan kembali mereview kembali **Arsitektur dari AlexNet** yang sudah kita bahas pada bagian sebelumnya. Dalam Paper tersebut, para penulis secara eksplisit menyebut: **"Another way to probe the network's visual knowledge is to consider the feature activations induced by an image at the last, 4096-dimensional hidden layer"**.



Jika kita perhatikan pada visualisasi Arsitektur Network AlexNet, *last hidden layer* adalah **FC7**. Layer **FC7** menghasilkan Vector berdimensi **4096**. Ini adalah *last hidden layer* yang menyimpan informasi "pemahaman" fitur secara menyeluruh sebelum data tersebut masuk ke lapisan klasifikasi (**FC8**). **FC8** adalah *Decision Layer* yang berisi 1000 neuron (sesuai **jumlah label pada Datasets ImageNet** https://www.image-net.org) yang bertugas untuk membuat keputusan, yaitu menebak label.

Catatan: FC = Fully Connected

Secara umum, Vector Embedding diambil sebelum *Output Decision Layer.*

Vector Embedding Part 7 | Melatih Language Model sederhana sebagai Vector Embedding Model

Sebelum menggunakan Model yang lebih powerful untuk menghasilkan Vector Embedding, untuk mendapatkan intuisi yang lebih baik dari mana Vector Embedding dihasilkan, kita akan **melatih Language Model sederhana** untuk kita jadikan sebagai Vector Embedding Model.

Cara kerja dari Model yang akan kita buat menggunakan mekanisme Training: Konteks vs Target.

Mekanisme Konteks vs Target ini menggunakan **BiGram**. Secara teknis, kita sedang mengajari model: *"Jika kata 'penyanyi' muncul, kemungkinan besar kata apa yang ada di dekatnya?"*. Output misalnya bisa: *“piano”, “gitar”, “vokalis”*.

- **X (Input):** One-Hot Encoding dari kata pusat.
- **Y (Output):** One-Hot Encoding dari kata tetangga.

Dimensi Vector Embedding dari model ini adalah 2. Artinya, kita memaksa model untuk merangkum seluruh makna kata ke dalam koordinat **2D (x, y)**.

Vector Embedding Part 8 | Melatih Language Model sederhana sebagai Vector Embedding Model

Dataset

Dataset untuk melatih model berupa artikel kecil kurang dari 50 baris.

“Anda bisa menambahkannya, Notebook dari Demo section ini akan disertakan di akhir section.”

```
import re
'''
cerita.txt
piano adalah alat musik
gitar termasuk alat musik
band memainkan musik
dalam band biasanya ada pemain gitar
dalam band biasanya ada pemain drum
vocalis biasanya ada dalam band
vocalis adalah penyanyi
vocalis biasanya nyanyi sambil memaninkan alat musik
penyanyi biasanya ada juga di dalam band
penyanyi solo biasanya bermain piano
penyanyi biasanya nyanyi sambil memaninkan alat musik
vocalis biasanya bisa memainkan piano
piano gitar dan drum biasanya ada dalam band
vocalis juga kadang bermain gitar
gamer bermain game
game biasanya ada di komputer
di playstation bisa bermain game
laptop dia ada game bagus
laptop, playstation dan komputer bisa digunakan untuk bermain game
nasi adalah makanan pokok
nasi cocok dengan tempe
tempe terbuat dari kedelai
kedelai adalah tanaman
sayur adalah tanaman
bayam adalah salah satu jenis sayur
'''

def remove_special_characters(text):
    pattern = r'[^a-zA-Z0-9\s]'  # Keep only alphanumeric characters and spaces
    return re.sub(pattern, '', text).strip()
```

Vector Embedding Part 9 | Melatih Language Model sederhana sebagai Vector Embedding Model

Membuat Dataset BiGram

Sesuai namanya, data **BiGram** terdiri dari 2 kata yang berurutan atau berdekatan.

Sebagai contoh, jika kita memiliki text:

Google adalah perusahaan teknologi dari Negara Amerika.

Setelah melalui data *cleansing* (misal menghilangkan *stopwords*). Data akan menjadi seperti berikut ini:

[“google”, “perusahaan”, “teknologi”, “negara”, “amerika”]

Kemudian proses **BiGram** akan menghasilkan data seperti berikut ini:

[[“google”, “perusahaan”], [“perusahaan”, “teknologi”], [“teknologi”, “negara”], [“negara”, “amerika”]]

Create BiGram

```
bigrams = []

for word_list in filtered_cerita_data:
  for i in range(len(word_list) - 1):
    for j in range(i+1, len(word_list)):
      bigrams.append([word_list[i], word_list[j]])
      bigrams.append([word_list[j], word_list[i]])

print(bigrams)
print(len(bigrams))
```

```
[['piano', 'alat'], ['alat', 'piano'], ['piano', 'musik'],
182
```

Vector Embedding Part 10 | Melatih Language Model sederhana sebagai Vector Embedding Model

Membangun Vocabulary

Seperti ***Language Model*** pada umumnya, kita juga membutuhkan daftar kosakata yang akan digunakan oleh Model yang akan kita bangun. *Vocabulary* dalam model ini berbentuk *simple* **Python Dictionary** untuk kebutuhan *lookup table*.

Get unique words

```
vocabs = []

for bigram in bigrams:
  vocabs.extend(bigram)

vocabs = list(set(vocabs))
vocabs.sort()
print(vocabs)
print(len(vocabs))
```

```
['alat', 'bagus', 'band', 'bayam', 'bermain', 'cocok', 'drum',
31
```

Create dictionary of words

```
vocab_dicts = {}

counter = 0
for v in vocabs:
  vocab_dicts[v] = counter
  counter = counter + 1

print(vocab_dicts)
```

```
{'alat': 0, 'bagus': 1, 'band': 2, 'bayam': 3, 'bermain': 4, '
```

Vector Embedding Part 11 | Melatih Language Model sederhana sebagai Vector Embedding Model

Merepresentasikan Vocabulary dalam bentuk One Hot Encoding/ Sparse Vector

Saat proses *training*, Model yang kita bangun membutuhkan input data numerik. Sehingga Data *training* harus kita ubah menjadi data numerik. Metode yang kita pilih adalah metode **One Hot Encoding/ Sparse Vector.** Anda bisa membaca kembali apa itu **Sparse Vector** pada bagian sebelumnya.

One Hot Encoding

```
import numpy as np

onehot_data = np.zeros((len(vocabs), len(vocabs)))

for i in range(len(vocabs)):
  onehot_data[i][i] = 1

onehot_data_dict = {}
c = 0
for v in vocabs:
  onehot_data_dict[v] = onehot_data[c]
  c = c + 1

print(onehot_data_dict)
```

```
{'alat': array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'bagus': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'band': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'bayam': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'bermain': array([0
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'cocok': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'drum': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'game': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'gamer': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'gitar': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'jenis': array([0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'kedelai': array([0
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'komputer': array([
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'laptop': array([0.
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'makanan': array([0
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'memainkan': array(
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'memaninkan': array
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'musik': array([0.,
```

Vector Embedding Part 12 | Melatih Language Model sederhana sebagai Vector Embedding Model

Mengimplementasikan Arsitektur Model menggunakan Pytorch

Model yang kita bangun bisa disebut *next word prediction* model. Model ini akan memprediksi kata berikutnya berdasarkan *input*. Jika melihat kembali Dataset **BiGram** yang kita buat sebelumnya, secara teknis kita sedang mengajari model: *“Jika kata penyanyi muncul, kemungkinan besar apa kata yang ada di dekatnya ?”* .

Input dan Output size model ini sama, yaitu jumlah keseluruhan **Vocabulary** yang kita buat sebelumnya. Untuk mempermudah penjelasan **Vector Embedding** dalam *section* materi ini dan **menghilangkan *step Dimensionality Reduction***, *hidden layer* kita buat menjadi 2 Neuron (lihat: **embed_size**). Kita paksa langsung *last hidden layer* merangkum seluruh **makna data** dalam ruang 2 Dimensi saja.

Train Model

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.autograd import Variable

# Get cpu, gpu or mps device for training.
device = ("cuda" if torch.cuda.is_available() else "mps" if torch.backends.mps.is

# Assuming X and Y are torch tensors
X = Variable(torch.tensor(x, dtype=torch.float32)).to(device)
Y = Variable(torch.tensor(y, dtype=torch.float32)).to(device)

# Define the neural network
class Net(nn.Module):
    def __init__(self, input_size, output_size, embed_size):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(input_size, embed_size)
        self.fc2 = nn.Linear(embed_size, output_size)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.softmax(self.fc2(x), dim=1)
        return x

    def get_embedded_vec(self, x):
      return torch.relu(self.fc1(x))

# Set seed for reproducibility
torch.manual_seed(42)

# Instantiate the model
embed_size = 2
model = Net(X.shape[1], Y.shape[1], embed_size).to(device)
```

Vector Embedding Part 13 | Melatih Language Model sederhana sebagai Vector Embedding Model

Validasi Model dengan mencoba memprediksi *next word* berdasarkan *input word*

Menggunakan input kata: “band” , model memprediksi outputnya dengan kata: “musik”.

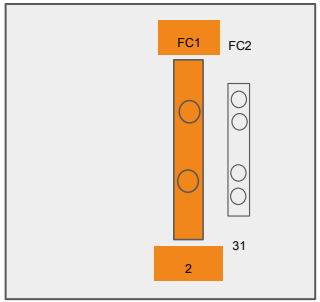
Tidak begitu buruk untuk Language Model kecil dengan 2 Network layer.

```
182
--------- embedded vec ---------
tensor([[4.1391, 0.0000]])
--------------------------------
tensor([[0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]])
band
tensor([[2.9531e-03, 4.5178e-15, 2.3838e-18, 2.9758e-15, 7.0277e-05, 3.9599e-14,
         1.2265e-16, 3.7772e-12, 3.8736e-16, 2.1167e-15, 1.0462e-15, 1.5366e-15,
         8.0372e-16, 3.7963e-15, 1.0503e-13, 8.3421e-17, 5.7419e-12, 9.9697e-01,
         5.6800e-20, 3.8787e-14, 2.3060e-15, 3.2904e-17, 7.1824e-18, 9.9821e-15,
         3.6623e-13, 1.7991e-16, 1.1058e-25, 2.7029e-16, 2.6132e-14, 2.2136e-06,
         1.6631e-17]])
musik
Predicted: "17"
/tmp/ipython-input-1682934348.py:15: UserWarning: Creating a tensor from a list of nu
  x = torch.tensor([x[index_data]], dtype=torch.float32)
```

Vector Embedding Part 14 | Melatih Language Model sederhana sebagai Vector Embedding Model

Mengkonversi Model menjadi Embedding Model

Mengkonversi Model menjadi Embedding Model dengan cara mengambil *last hidden layer* dari model. Dengan jumlah 2 Neuron pada last hidden layer, Vector Embedding yang dihasilkan oleh model ini akan memiliki dimensi 2 Dimensi.



FC1 adalah *last hidden layer* sebelum *decision layer,* yaitu FC2. Layer ini yang kita gunakan untuk menghasilkan **Vector Embedding**.

```
model = Net(X.shape[1], Y.shape[1], embed_size).to(device)
print(model)
model.load_state_dict(torch.load("model.pth"))

weights = model.fc1.weight.detach().cpu().numpy()

word_embeddings = {}
for v in vocabs:
  word_embeddings[v] = [weights[0][vocab_dicts[v]], weights[1][vocab_dicts[v]]]
word_embeddings
```

```
Net(
  (fc1): Linear(in_features=31, out_features=2, bias=True)
  (fc2): Linear(in_features=2, out_features=31, bias=True)
)
```

Vector Embedding Part 15 | Melatih Language Model sederhana sebagai Vector Embedding Model

Normalisasi Vector dan melakukan Vector Search dengan Cosine Similarity

Sebelum melakukan **Vector Search**, kita terlebih dahulu menormalisasi seluruh **Vector Embedding** yang merepresentasikan setiap kata yang ada di *Vocabulary* kita. Jika kita ingat pada pembahasan sebelumnya, **Cosine Similarity** hanya butuh arah (*direction*), tidak membutuhkan besaran/panjang (*magnitude*).

Pada pencarian dengan input kata: **“penyanyi”** dan **top_k=5**, **Vector Search** berhasil mendapatkan 5 kata dengan koordinat terdekat. Yaitu *“vocalis”, “piano”, “alat”, “band” dan “nyanyi”*.

Vector Normalization

```
def normalize(v):
    v = np.array(v, dtype=float)
    return v / np.linalg.norm(v)

normalized_embeddings = {
    w: normalize(vec)
    for w, vec in word_embeddings.items()
}

normalized_embeddings
```

```
def cosine_sim_norm(a, b):
    return np.dot(a, b)

print(cosine_sim_norm(
    normalized_embeddings["piano"],
    normalized_embeddings["gitar"]
))
```

```
-0.03638797060264176
```

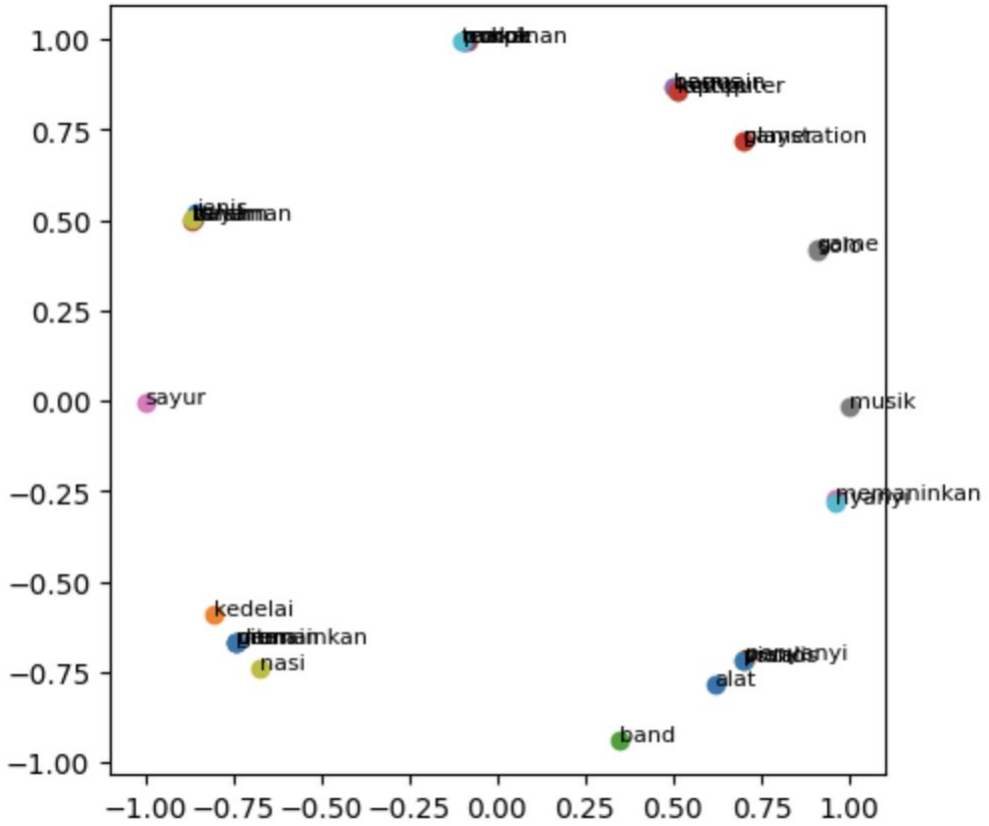
```
def most_similar(word, embeddings, top_k=5):
    target = embeddings[word]
    scores = []

    for w, vec in embeddings.items():
        if w == word:
            continue
        sim = cosine_sim_norm(target, vec)
        scores.append((w, sim))

    scores.sort(key=lambda x: x[1], reverse=True)
    return scores[:top_k]

similar_results = most_similar("penyanyi", normalized_embeddings)
for word, score in similar_results:
    print(f"{word}: {score}")
```

```
vocalis: 0.9999753137323064
piano: 0.9999722660228834
alat: 0.9938282555467914
band: 0.911491556365883
nyanyi: 0.8737595036631776
```

Vector Embedding Part 16 | Melatih Language Model sederhana sebagai Vector Embedding Model

Memvisualisasikan seluruh Vocabulary dalam ruang 2 Dimensi

Pada ruang 2 Dimensi, **Vector Embedding** yang dihasilkan oleh **Language Model** kecil yang kita buat berhasil menempatkan kata yang saling berkaitan satu sama lain pada koordinat yang berdekatan. Misalnya kedelai dan nasi, playstation dan game.



Vector Embedding Part 17 | Melatih Language Model sederhana sebagai Vector Embedding Model

URL Notebook Melatih NLP Model sederhana sebagai Vector Embedding Model

https://colab.research.google.com/drive/1oTlaXgx9rTm3t0SxAuijMJnEZYk0_NB5?usp=sharing

Anda bisa bereksperimen sendiri dengan Notebook ini, untuk mendapatkan intuisi yang lebih baik tentang darimana Vector Embedding dihasilkan.

Vector Embedding Part 18 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Menghasilkan Sentence Embedding menggunakan "sentence-transformers/paraphrase-multilingual-mpnet-base-v2"

Model **sentence-transformers/paraphrase-multilingual-mpnet-base-v2** adalah salah satu **Embedding Model** yang dikembangkan oleh tim **Sentence Transformers (SBERT)** yang diperuntukan menghasilkan **Vector Embedding** untuk kebutuhan **Sentence Similarity**, **Text Similarity** dan **Semantic Text Search**.

Model ini dilatih khusus untuk memahami bahwa dua kalimat dengan kata-kata berbeda bisa memiliki makna yang sama (*paraphrase*).
Model ini menghasilkan **Vector Embedding** dengan **ukuran 768 Dimensi**.

```python
from sentence_transformers import (
    SentenceTransformer,
    util
)

sentence_1 = "saya sedang tidak enak badan"

model = SentenceTransformer('sentence-transformers/paraphrase-multilingual-mpnet-base-v2')

embedding_1 = model.encode(sentence_1)

print(embedding_1[:10])
```

```
[ 0.01320906 -0.00484423 -0.01073376  0.03327633  0.08287398  0.03472598
 -0.07229888  0.06802528  0.1447741   0.07492868]
```

Menampilkan **10 value pertama** dari Vector Embedding.

Vector Embedding Part 19 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Mengukur Sentence Similarity Vector Embedding yang dihasilkan “sentence-transformers/paraphrase-multilingual-mpnet-base-v2”

Perhatikan bahwa hasil terbaik adalah *"saya sedang tidak enak badan"* dan *"saya merasa kesehatan saya sedang terganggu"*. Keduanya tidak memiliki kata "menurun" atau "fisik", tapi secara koordinat Vector dan menggunajan Metric Cosine Similarity, mereka adalah **tetangga terdekat (Nearest Neighbors)**. “**sentence-transformers/paraphrase-multilingual-mpnet-base-v2**” sangat baik dalam memahami “jika” dua kalimat memiliki makna (*paraphrase*) yang sama.

```python
from sentence_transformers import SentenceTransformer, util
import torch

model = SentenceTransformer('sentence-transformers/paraphrase-multilingual-mpnet-base-v2')

# in memory database
documents = [
    "saya sedang tidak enak badan",
    "hari ini cuaca sangat cerah sekali",
    "laptop saya sedang rusak dan perlu diperbaiki",
    "saya merasa kesehatan saya sedang terganggu"
]

# indexing
doc_embeddings = model.encode(documents)

# input query
query = "saya sedang sakit"
query_embedding = model.encode(query)

# vector search with cosine similarity
cosine_scores = util.cos_sim(query_embedding, doc_embeddings)[0]

# search the top k
top_results = torch.topk(cosine_scores, k=2)

print(f"User Query: '{query}'\n")
print("Top 2 Search Results in Vector Database:")
print("-" * 40)

for score, idx in zip(top_results.values, top_results.indices):
    print(f"Score: {score:.4f}")
    print(f"Document: {documents[idx]}")
    print("-" * 40)
```

```
User Query: 'saya sedang sakit'

Top 2 Search Results in Vector Database:
----------------------------------------
Score: 0.7781
Document: saya sedang tidak enak badan
----------------------------------------
Score: 0.6258
Document: saya merasa kesehatan saya sedang terganggu
----------------------------------------
```

Vector Embedding Part 20 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Menghasilkan Image Embedding menggunakan “facebook/dinov2-small”

Model **facebook/dinov2-small** adalah salah satu **Embedding Model** berdasarkan Arsitektur ViT (Vision Transformers) yang dikembangkan oleh tim **Meta** dan diperuntukan untuk menghasilkan Vector Embedding dari gambar (*image*). Kemudian hasil Vector Embedding Image bisa kita gunakan untuk *task* lebih lanjut. Misalnya **image to image search** atau bahkan melatih **zero shot image classification**.

Sesuai namanya **“facebook/dinov2-small”**, model ini menghasilkan Vector Embedding dengan **ukuran 384 Dimensi**. **Untuk level produksi, anda mungkin perlu menggunakan varian Model DINO yang lebih besar.**

```python
import os
import numpy as np
import requests
from PIL import Image

import torch
from transformers import AutoImageProcessor, AutoModel

device = "cuda" if torch.cuda.is_available() else "cpu"

EMBEDDING_MODEL = "facebook/dinov2-small"
processor = AutoImageProcessor.from_pretrained(EMBEDDING_MODEL)
model = AutoModel.from_pretrained(EMBEDDING_MODEL).to(device)
model.eval()

def get_embedding(img_url):
    image = Image.open(requests.get(img_url, stream=True).raw).convert("RGB")
    inputs = processor(images=image, return_tensors="pt").to(device)
    with torch.no_grad():
        outputs = model(**inputs)
        embedding = outputs.last_hidden_state[:, 0, :]
        embedding = torch.nn.functional.normalize(embedding, dim=-1)  # normalize optional
    return embedding.squeeze().cpu().numpy()

img_url_1 = "https://images-na.ssl-images-amazon.com/images/I/41u48iJce7L.jpg"

print(get_embedding(img_url_1)[:10])
```

```
[-0.10202865 -0.01010288 -0.02814776  0.05421898  0.00209667 -0.04328454
  0.04019937  0.03087261 -0.0392565  -0.12512551]
```

Menampilkan **10 value pertama** dari Vector Embedding.

Vector Embedding Part 21 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Mengukur Image Similarity Vector Embedding yang dihasilkan “facebook/dinov2-small”

```python
import os
import numpy as np
import requests
from PIL import Image

import torch
from transformers import AutoImageProcessor, AutoModel

device = "cuda" if torch.cuda.is_available() else "cpu"

EMBEDDING_MODEL = "facebook/dinov2-small"
processor = AutoImageProcessor.from_pretrained(EMBEDDING_MODEL)
model = AutoModel.from_pretrained(EMBEDDING_MODEL).to(device)
model.eval()

def get_embedding(img_url):
    image = Image.open(requests.get(img_url, stream=True).raw).convert("RGB")
    inputs = processor(images=image, return_tensors="pt").to(device)
    with torch.no_grad():
        outputs = model(**inputs)
        embedding = outputs.last_hidden_state[:, 0, :]
        embedding = torch.nn.functional.normalize(embedding, dim=-1)  # normalize optional
    return embedding.squeeze().cpu().numpy()

img_url_1 = "https://images-na.ssl-images-amazon.com/images/I/41u48iJce7L.jpg"
img_url_2 = "https://images-na.ssl-images-amazon.com/images/I/41wqBSDd2AL.jpg"
img_url_3 = "https://images-na.ssl-images-amazon.com/images/I/41tueGfkp6L.jpg"

img_url_4 = "https://unsplash.com/photos/X0mnZn264uA/download?force=true&w=640"
img_url_5 = "https://unsplash.com/photos/1HqixV1agUw/download?force=true&w=640"
img_url_6 = "https://unsplash.com/photos/EGzkhZyFRX4/download?force=true&w=640"

input_image = "https://unsplash.com/photos/EEddCahUvY8/download?force=true&w=640"
```

```python
# vector search
import matplotlib.pyplot as plt

# 1. Database
image_library = [img_url_1, img_url_2, img_url_3, img_url_4, img_url_5]

# indexing
print("Indexing image...")
library_embeddings = np.array([get_embedding(url) for url in image_library])

print(f"Searching image...")
query_vec = get_embedding(input_image)

# search with Cosine Similarity
scores = np.dot(library_embeddings, query_vec)

# get the top 3 results
top_k = 3
top_indices = np.argsort(scores)[::-1][:top_k]

fig = plt.figure(figsize=(15, 5))

# show Input Image
ax_input = fig.add_subplot(1, top_k + 1, 1)
img_input = Image.open(requests.get(input_image, stream=True).raw)
ax_input.imshow(img_input)
ax_input.set_title("INPUT IMAGE\n(Query)", color='blue', fontweight='bold')
ax_input.axis('off')

# show Top-K Results
for i, idx in enumerate(top_indices):
    ax_res = fig.add_subplot(1, top_k + 1, i + 2)
    img_res = Image.open(requests.get(image_library[idx], stream=True).raw)
    ax_res.imshow(img_res)
    ax_res.set_title(f"Rank {i+1}\nScore: {scores[idx]:.4f}")
    ax_res.axis('off')

plt.tight_layout()
plt.show()
```

Simulasi **image to image search** menggunakan model **“facebook/dinov2-small”**. Secara umum step-step yang dilakukan adalah *data indexing* (**ada proses Image Embedding**), simpan data ke Vector Store (dalam simulasi ini **in memory** pada **Python List**), Image Vector Search (ada proses **Image Embedding sebelum Query**).

Vector Embedding Part 22 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Mengukur Image Similarity Vector Embedding yang dihasilkan “facebook/dinov2-small”

Hasil Simulasi **image to image search** menggunakan model **“facebook/dinov2-small”** menunjukan model ini bisa sangat baik dalam membandingkan **Semantic Visual** pada satu gambar dengan gambar lainnya, meskipun secara struktur pixel, latar belakang, warna, bahkan dengan bentuk yang berbeda.

Vector Embedding Part 23 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Mengukur Text to Image Similarity atau Image to Image Similarity dan menghasilkan Vector Embedding menggunakan Multimodal Model "sentence-transformers/clip-ViT-B-32-multilingual-v1"

"sentence-transformers/clip-ViT-B-32-multilingual-v1" adalah implementasi dari CLIP (*Contrastive Language–Image Pre-training*) milik OpenAi yang dikembangkan oleh tim **Sentence Transformers (SBERT)**. Model ini dilatih untuk memahami hubungan antara teks dan gambar dalam satu ruang Vector yang sama. Memungkin kita mencari kemiripan antara text (yang mendeskripsikan sebuah visual) dengan gambar.

Misalnya anda memasukan *query*: "Minum kopi sepertinya enak", kemudian model menghasilkan **Vector Embedding** representasi dari *query* tersebut. Selanjutnya **Vector Embedding** tersebut digunakan sebagai input untuk **Vector Search**. Menggunakan *metrics* "misalnya" **Cosine Similarity**, hasil mungkin akan berupa gambar orang sedang membuat kopi, gambar kopi di kebun kopi, gambar orang minum kopi di cafe.

Vector Embedding Part 24 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Mengukur Text to Image Similarity atau Image to Image Similarity dan menghasilkan Vector Embedding menggunakan Multimodal Model "sentence-transformers/clip-ViT-B-32-multilingual-v1"

```python
from sentence_transformers import SentenceTransformer, util
from PIL import Image, ImageFile
import requests
import torch

img_model = SentenceTransformer('clip-ViT-B-32')

text_model = SentenceTransformer('sentence-transformers/clip-ViT-B-32-multilingual-v1')


def load_image(url_or_path):
    if url_or_path.startswith("http://") or url_or_path.startswith("https://"):
        return Image.open(requests.get(url_or_path, stream=True).raw)
    else:
        return Image.open(url_or_path)

img_url_1 = "https://images-na.ssl-images-amazon.com/images/I/41u48iJce7L.jpg"
img_url_2 = "https://images-na.ssl-images-amazon.com/images/I/41wqBSDd2AL.jpg"
img_url_3 = "https://images-na.ssl-images-amazon.com/images/I/41tueGfkp6L.jpg"

img_url_4 = "https://unsplash.com/photos/X0mnZn264uA/download?force=true&w=640"
img_url_5 = "https://unsplash.com/photos/1HqixV1agUw/download?force=true&w=640"
img_url_6 = "https://unsplash.com/photos/EGzkhZyFRX4/download?force=true&w=640"

input_text_query = "naik atv motor sepertinya enak"
```

```python
# vector search
import numpy as np
import matplotlib.pyplot as plt

image_library = [img_url_1, img_url_2, img_url_3, img_url_4, img_url_5, img_url_6]

# Indexing Image
print("Indexing images into Shared Vector Space...")

images_pil = [load_image(url) for url in image_library]
library_embeddings = img_model.encode(images_pil)

# Query
print(f"Searching images for query: '{input_text_query}'...")
query_vec = text_model.encode(input_text_query)

# 4. Search with Cosine Similarity
scores = np.dot(library_embeddings, query_vec)

# Ranking
top_k = 3
top_indices = np.argsort(scores)[::-1][:top_k]

fig = plt.figure(figsize=(15, 5))

# show text query
ax_text = fig.add_subplot(1, top_k + 1, 1)
ax_text.text(0.5, 0.5, f"QUERY:\n\n'{input_text_query}'",
             fontsize=12, ha='center', va='center', fontweight='bold', color='blue')
ax_text.set_title("INPUT TEXT", fontweight='bold')
ax_text.axis('off')

# show Top-K Image Results
for i, idx in enumerate(top_indices):
    ax_res = fig.add_subplot(1, top_k + 1, i + 2)
    img_res = images_pil[idx]
    ax_res.imshow(img_res)
    ax_res.set_title(f"Rank {i+1}\nScore: {scores[idx]:.4f}")
    ax_res.axis('off')

plt.tight_layout()
plt.show()
```

Simulasi **Multimodal search** menggunakan model **"sentence-transformers/clip-ViT-B-32-multilingual-v1**. Secara umum step-step yang dilakukan adalah *data indexing* (**ada proses Image Embedding**), simpan data ke Vector Store (dalam simulasi ini **in memory** pada **Python List**), Image/Text Vector Search (ada proses **Image Embedding sebelum Query**).

Vector Embedding Part 25 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

Mengukur Text to Image Similarity atau Image to Image Similarity dan menghasilkan Vector Embedding menggunakan Multimodal Model "sentence-transformers/clip-ViT-B-32-multilingual-v1"

Hasil Simulasi **Multimodal search** menggunakan model **"sentence-transformers/clip-ViT-B-32-multilingual-v1"** menunjukan model ini bisa sangat baik dalam membandingkan **text yang mendeskripsikan sebuah visual dengan visual pada gambar**.

Vector Embedding Part 26 | Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

URL Notebook Menggunakan Pre-trained Embedding Model untuk menghasilkan Vector Embedding

https://colab.research.google.com/drive/1uQlxkB5wJFzNmj2elg5lFxC7kBEE7Sma?usp=sharing

Anda bisa bereksperimen sendiri dengan Notebook ini, untuk mendapatkan intuisi bagaimana cara menghasilkan Vector Embedding dari pre-trained model.

Vector Database

Vector Database Part 1

Bayangkan kita memiliki jutaan foto dan dokumen, namun semuanya tidak memiliki nama file. Database tradisional akan sulit pada beberapa kasus ini. Di sinilah **Vector Database** berperan. **Ia tidak menyimpan data berdasarkan 'label' atau 'nama', melainkan berdasarkan 'posisi' atau 'makna' dalam ruang multidimensi**. Dengan teknologi ini, kita tidak lagi bertanya kepada komputer *'Cari dokumen yang mengandung kata X'*, melainkan *'Cari semua data yang memiliki nuansa, feels atau bentuk yang mirip dengan input saya'*. Inilah fondasi utama yang memungkinkan AI memiliki *memory* yang terstruktur dan sangat cepat.

Beberapa alasan kita membutuhkan Vector Database, dibandingkan kita menyimpan Vector Embedding pada in memory di level code, misalnya List atau Dictionary.

- **Penyimpanan Efektif:** Dioptimalkan khusus untuk menyimpan *Dense Vectors* hasil dari Embedding Model CLIP, DINO, MPNet, Open Ai Embedding atau Embedding Model Lain.
- **Indexing Cepat:** Menggunakan algoritma **ANN (Approximate Nearest Neighbors)** seperti **HNSW** atau **IVFFlat** untuk proses pencarian. Jika kita menggunakan metode tradisional atau bahkan melakukan *sort* manual, cara ini sangat tidak *scalable*. Sebab, metode tradisional atau *sort* manual melakukan perbandingan satu persatu ke setiap *row* Vector Embedding yang ada.
- **Kebutuhan Memory Jangka Panjang:** Jika anda pernah mendengar atau bahkan membangun langsung aplikasi yang bertumpu pada ***Retrieval-Augmented Generation*** **(RAG)** seperti Chatbot yang memanfaatkan LLM (*Large Language Model)*, anda membutuhkan *memory* jangka panjang untuk menyimpan konteks yang akan anda *lookup* berdasarkan *prompt/query* dari pengguna.

Vector Database Part 2

Ada banyak **Database Vendor** yang menyediakan/mengimplementasikan **Vector Store** untuk kebutuhan **Vector Search**. Kita akan mencoba menggunakan **3 Vector Store** yang secara umum mewakili beberapa kebutuhan dan kondisi infrastruktur dalam suatu organisasi. Yaitu **Postgre(ext: pgvector)**, **Elasticsearch**, dan **Qdrant**.

PostgreSQL (pgvector) untuk solusi All-in-One

- **Karakteristik:** Ekstensi dari RDBMS paling populer di dunia.
- **Mengapa menggunakan ini?** Cocok untuk aplikasi yang sebelumnya sudah menggunakan PostgreSQL, sebab kita tidak membutuhkan infrastruktur baru khusus untuk Vector Store (instalasi terpisah).
- **Kelebihan:** Mendukung transaksi ACID, sangat stabil, dan sekarang sudah mendukung algoritma Approximate Nearest Neighbors (ANN) **HNSW** dan **IVFFlat** yang digunakan langsung sebagai mekanisme Indexing.

https://www.postgresql.org/

https://github.com/pgvector/pgvector

Vector Database Part 3

PostgreSQL (pgvector) untuk solusi All-in-One

Memasang ekstensi pgvector:

Sebelum melakukan kompilasi, sistem memerlukan paket pengembangan (*development headers*) yang sesuai dengan versi PostgreSQL yang digunakan. Dalam hal ini, kita menggunakan PostgreSQL versi 14.

```
sudo apt-get update && sudo apt-get install -y postgresql-server-dev-14 build-essential git
```

Berpindah ke direktori sementara:

```
cd /tmp
```

Melakukan kloning repositori pgvector dengan branch spesifik v0.8.1:

```
git clone --branch v0.8.1 https://github.com/pgvector/pgvector.git
```

Masuk ke direktori repositori:

```
cd pgvector
```

Membersihkan residu kompilasi sebelumnya (jika ada) :

```
make clean
```

Tahap Kompilasi source code. Penggunaan OPTFLAGS="" memastikan kompatibilitas instruksi CPU yang lebih luas:

```
make OPTFLAGS=""
```

Menginstal binary ekstensi ke direktori library PostgreSQL :

```
make install
```

Vector Database Part 4

PostgreSQL (pgvector) untuk solusi All-in-One

Mengaktifkan ekstensi Vector

Perintah SQL untuk mengaktifkan fitur Vector:

```
CREATE EXTENSION IF NOT EXISTS vector;
```

Verifikasi instalasi dan versi ekstensi:

```
SELECT extname, extversion FROM pg_extension WHERE extname = 'vector';
```

Membuat tabel contoh: document_embeddings dengan kolom embedding kita set dimensinya: 768, sesuai ukuran *output* Embedding model yang akan kita gunakan, yaitu “sentence-transformers/paraphrase-multilingual-mpnet-base-v2”.

```
CREATE TABLE document_embeddings (
  id serial PRIMARY KEY,
  title TEXT,
  content TEXT,
  source TEXT,
  embedding vector(768),  -- Assuming you're using a 768-dim embedding
  last_updated TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Membuat Index menggunakan dengan HNSW:

```
CREATE INDEX ON document_embeddings USING hnsw (embedding vector_cosine_ops);
```

Vector Database Part 5

PostgreSQL (pgvector) untuk solusi All-in-One

Data ingestion dari dokumen seperti PDF, Docx, dan PPTx file

```python
def insert_embedding(pg, content, source, model):
    # Generate embedding
    embedding = model.encode(content).tolist()

    # Get the current timestamp
    last_updated = datetime.now()

    # Prepare SQL query
    sql = """
    INSERT INTO document_embeddings (content, source, embedding, last_updated)
    VALUES (%s, %s, %s, %s)
    """

    # Insert the document content, source, and embedding into the table
    pg.execute_and_commit(sql, (content, source, embedding, last_updated))

if __name__ == '__main__':
    model = SentenceTransformer('./model/paraphrase-multilingual-mpnet-base-v2')

    pg = postgres.PostgresSync(conn_str='postgresql://user:12345678@127.0.0.1:5432/mydb')
    try:
       pg.connect()
    except Exception as e:
       print(e)

    curr_dir = os.getcwd()

    files = []
    for f in os.listdir(os.path.join(curr_dir, 'docs')):
        if os.path.isfile(os.path.join(curr_dir, f'docs/{f}')):
            files.append(os.path.join(curr_dir, f'docs/{f}'))

    for file_path in files:
        chunks = doc_util.process_document(file_path)
        for chunk in chunks:
            insert_embedding(pg, chunk['content'], chunk['source'], model)

    # disconnect db
    pg.disconnect()
```

Vector Database Part 6

PostgreSQL (pgvector) untuk solusi All-in-One

Melakukan Vector Search menggunakan metrics Cosine Similarity

```python
@app.route("/search-doc")
def search_doc():
    def query_similar_documents(pg, query, k=2):
        # Generate the embedding for the query
        query_embedding = embed_text(query).tolist()
        query_embedding_str = f"{query_embedding}"

        # Query for the top-k most similar documents using cosine similarity
        sql = """
        SELECT id, content, source, 1 - (embedding <=> %s) AS score,
        last_updated
        FROM document_embeddings
        ORDER BY score DESC
        LIMIT %s
        """
        results = pg.execute_query(sql, (query_embedding_str, k))
        return results

    q = request.args.get('q')
    k = request.args.get('k')
    if q is None or q == '':
        return jsonify({'message': 'search query cannot be empty'}), 400

    if k is None or k == '':
        k = '2'

    columns = (
        'id', 'content', 'source', 'score', 'last_updated'
    )

    rows = query_similar_documents(pg, q, k=int(k))

    results = []
    for row in rows:
        results.append(dict(zip(columns, row)))

    return jsonify({
        'data': results,
        'message': 'search result'
    })
```

Pgvector mendukung distance function:

- <-> - L2 distance
- <#> - (negative) inner product
- <=> - cosine distance
- <+> - L1 distance
- <~> - Hamming distance (binary vectors)
- <%> - Jaccard distance (binary vectors)

Pgvector *by default* tidak menyediakan fungsi **Cosine Similarity,** untuk menggunakan fungsi Cosine Similarity, cukup gunakan **trik**:

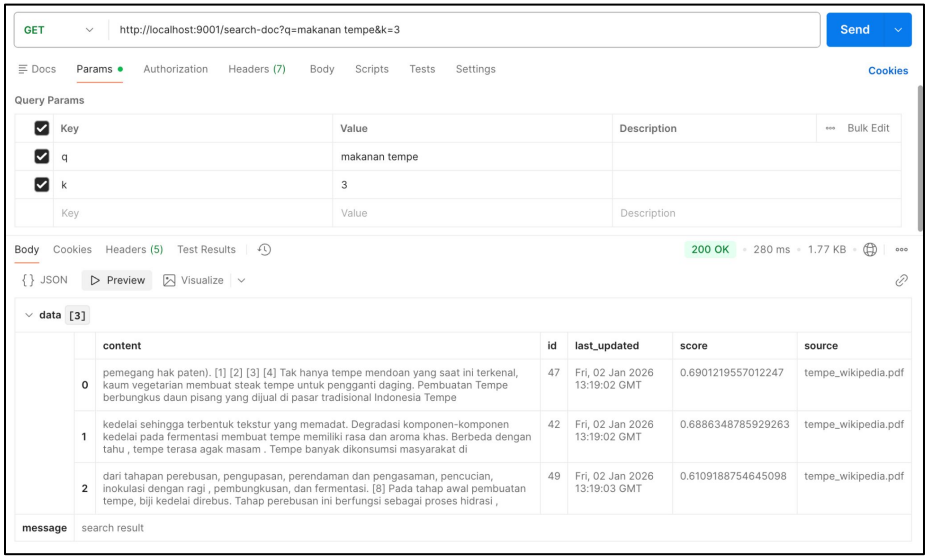
Cosine Similarity = 1 - (Cosine Distance)

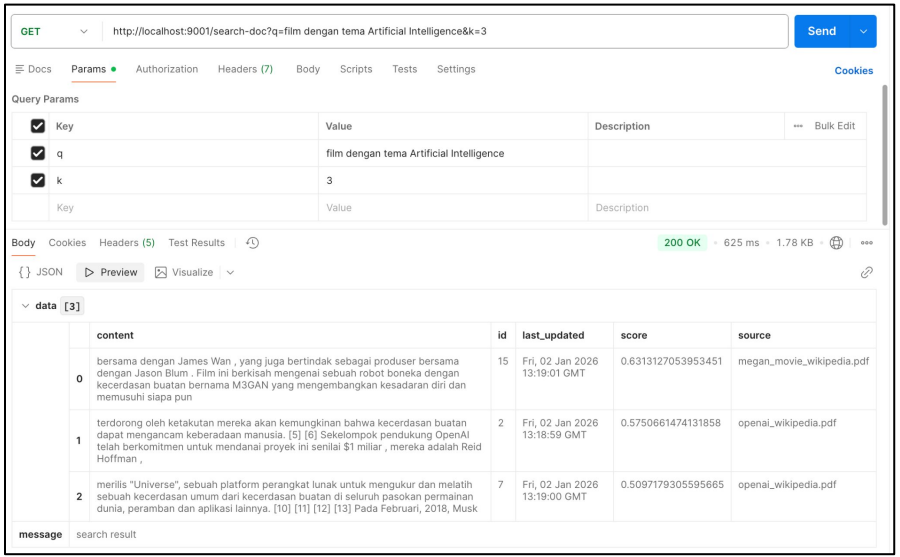
Vector Database Part 7

PostgreSQL (pgvector) untuk solusi All-in-One

Melakukan Vector Search menggunakan metrics Cosine Similarity

Menampilkan hasil pencarian dokumen dengan query tertentu.





Vector Database Part 8

Qdrant untuk kebutuhan Native Vector Store

- **Karakteristik:** Database yang dirancang khusus dari awal untuk mengelola data Vector (*Vector-Native*).
- **Implementasi:** Pilihan utama untuk kebutuhan dengan prioritas tinggi pada performa dan efisiensi. Dibangun menggunakan bahasa pemrograman **Rust**, **Qdrant** menawarkan kecepatan pemrosesan tinggi dan kemudahan integrasi melalui API.
- **Kelebihan:** Memiliki kemampuan *advanced filtering*) yang sangat presisi dan cepat, serta manajemen memori yang optimal untuk Dataset skala besar.

https://qdrant.tech/

Vector Database Part 9

Qdrant untuk kebutuhan Native Vector Store

Data ingestion dari dokumen seperti PDF, Docx, dan PPTx file

```python
from logger import logger as log
from qdrant_client import models, QdrantClient
from sentence_transformers import (
    SentenceTransformer
)

class QDrantDb:
    def __init__(self, conn_str: str, dense_model: SentenceTransformer) -> None:
        self.client = None
        self.database_url = conn_str
        self.dense_model = dense_model

    def connect(self):
        self.client = QdrantClient(url=self.database_url)

    def create_collections(self):
        if self.client is None:
            raise Exception('client not initialized yet')

        if not self.client.collection_exists("document_embeddings"):
            self.client.create_collection(
                collection_name="document_embeddings",
                vectors_config=models.VectorParams(
                    size=self.dense_model.get_sentence_embedding_dimension(),  # Vector size is defined by used model
                    distance=models.Distance.COSINE,
                )
            )
            log.info('create collections document_embeddings succeed')

    def disconnect(self):
        if self.client is not None:
            self.client.close()
```

```python
def insert_embedding(qdrant_db: qdrantdb.QDrantDb, chunks):
    qdrant_db.client.upsert(
        collection_name="document_embeddings",
        points=[
            qdrantdb.models.PointStruct(
                id=uuid.uuid4().hex,
                vector=qdrant_db.dense_model.encode(doc['content']).tolist(),
                payload={'content': doc['content'], 'source': doc['source'], 'last_updated': datetime.now()}
            )
            for doc in chunks
        ]
    )

if __name__ == '__main__':
    model = SentenceTransformer('./model/paraphrase-multilingual-mpnet-base-v2')

    qdrant_db = qdrantdb.QDrantDb(conn_str='http://localhost:6333', dense_model=model)
    try:
        # connect
        qdrant_db.connect()

        # create collections
        qdrant_db.create_collections()
    except Exception as e:
        print('qdrant connect errors')
        print(e)

    curr_dir = os.getcwd()

    files = []
    for f in os.listdir(os.path.join(curr_dir, 'docs')):
        if os.path.isfile(os.path.join(curr_dir, f'docs/{f}')):
            files.append(os.path.join(curr_dir, f'docs/{f}'))

    for file_path in files:
        chunks = doc_util.process_document(file_path)
        insert_embedding(qdrant_db=qdrant_db, chunks=chunks)

    # disconnect db
    qdrant_db.disconnect()
```

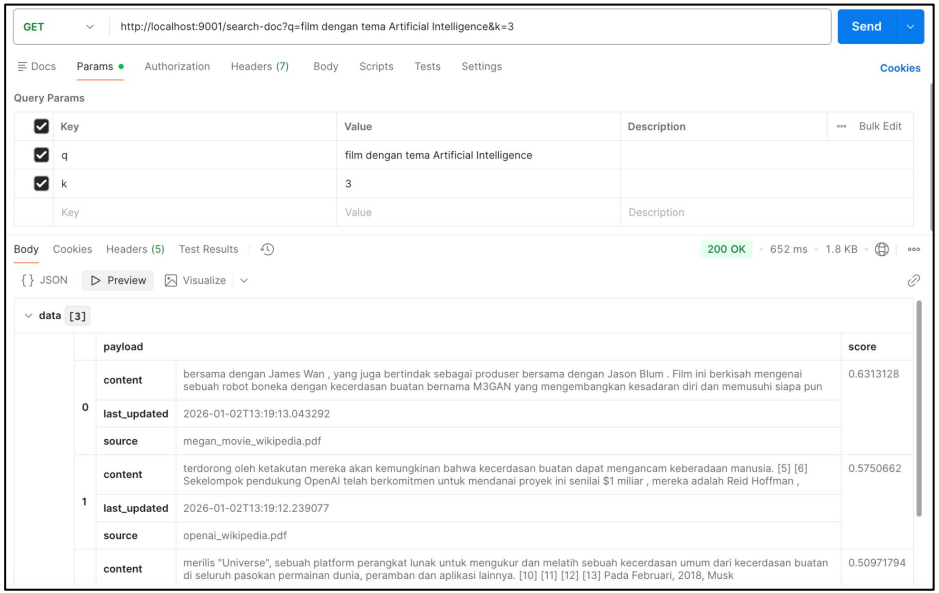
Insight penting: saat membuat collection di Qdrant, kita sekaligus menentukan Embedding Dimension dan Distance function yang akan kita gunakan. Ukuran Embedding Dimension = 768, sesuai ukuran *output* Embedding model yang akan kita gunakan, yaitu “sentence-transformers/paraphrase-multilingual-mpnet-base-v2”

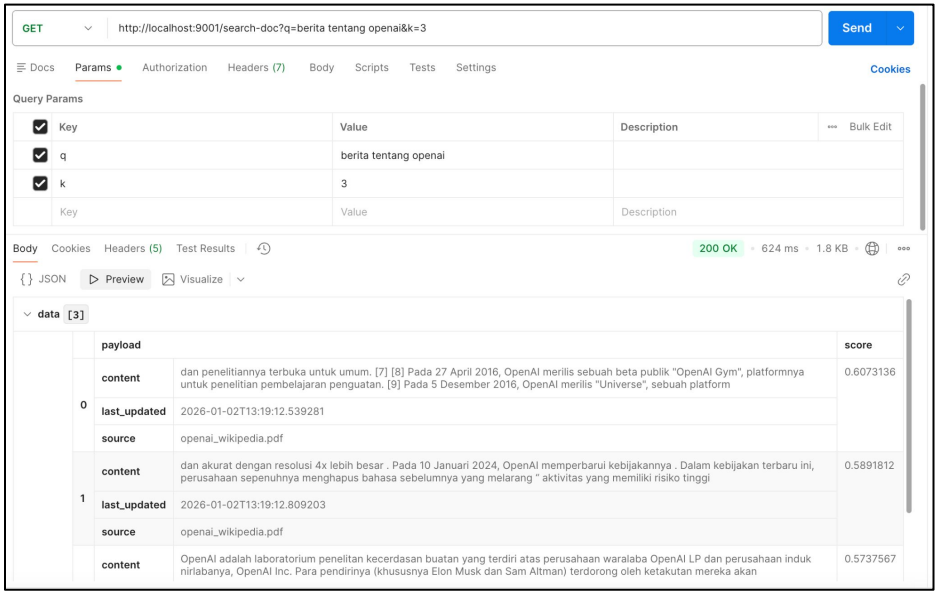
Vector Database Part 10

Qdrant untuk kebutuhan Native Vector Store

Melakukan Vector Search menggunakan metrics Cosine Similarity

Menampilkan hasil pencarian dokumen dengan query tertentu.





Vector Database Part 11

Elasticsearch untuk solusi Hybrid Search

- **Karakteristik:** Standar industri untuk ***full text search*** yang kini mendukung *Dense Vector Search*.
- **Implementasi:** Sangat direkomendasikan untuk skenario **Hybrid Search**. Memungkinkan integrasi antara pencarian kata kunci tradisional (*keyword-based search*) dengan *semantic search* berbasis makna (*vector-based search*) secara simultan.
- **Kelebihan:** Memiliki skalabilitas tinggi untuk *volume* data masif, serta dilengkapi dengan fitur analitik dan pemantauan data yang sangat komprehensif. Selain itu mirip dengan kasus PostgreSQL. Ketika kita sudah memiliki infrastruktur untuk Elasticsearch, kita tidak perlu lagi membutuhkan infrastruktur baru yang dikhususkan untuk pemasangan Vector Store terpisah.

https://www.elastic.co/docs/solutions/search/vector

https://www.elastic.co/search-labs/blog/vector-search-set-up-elasticsearch

Vector Database Part 12

Elasticsearch untuk solusi Hybrid Search

Melakukan Vector Search image to image search menggunakan Model sentence-transformers/clip-ViT-B-32 dan metrics Cosine Similarity

Field Mapping untuk field imageVector

```
"imageVector": {
  "type": "dense_vector",
  "dims": 512,
  "index": true,
  "similarity": "cosine",
  "index_options": {
    "type": "int8_hnsw",
    "ef_construction": 128,
    "m": 24
  }
},
```

Field Mapping untuk field **imageVector**

- **Tipe Data (dense_vector)**: Field ini dikonfigurasi khusus untuk menyimpan *high-dimensional embeddings* yang dihasilkan oleh model AI.
- **Dimensi (dims: 512)**: Konfigurasi ini disamakan dengan dimensi *output* dari model **CLIP-ViT-B-32**, memastikan integritas data pada saat proses *indexing*.
- **Metrics (cosine)**: Menggunakan *Cosine Similarity* sebagai Vector metrics. Disesuaikan dengan model **CLIP,** karena orientasi Vector lebih menentukan makna *semantic* dibandingkan besaran/panjang (magnitude).
- Untuk kebutuhan Approximate Nearest Neighbors, Kita menggunakan **Scalar Quantization (int8)**. Teknik ini mengonversi Vector dari format float32 ke int8, yang secara drastis bisa mengurangi konsumsi Memori (RAM).

Parameter Indexing (HNSW) https://www.elastic.co/search-labs/blog/hnsw-graph

- m = 24.
- ef_construction = 128

Vector Database Part 13

Elasticsearch untuk solusi Hybrid Search

Melakukan Vector Search image to image search menggunakan Model sentence-transformers/clip-ViT-B-32 dan metrics Cosine Similarity

Data Ingestion dari file gambar

```python
def ingest():
    # https://huggingface.co/sentence-transformers/clip-ViT-B-32
    model = SentenceTransformer('clip-ViT-B-32')

    esclient = Elasticsearch(
        hosts='http://127.0.0.1:9200',
        basic_auth=('elastic', 'changeme'),
        request_timeout=30,
        max_retries=3,
        retry_on_timeout=True
    )

    print(esclient.info())

    df = pd.read_csv('../amazon_products.csv')
    df = df.reset_index()

    index_name = 'products'
    def bulk_insert():
        for index, row in df.iterrows():
            productName = row['Product Name']
            category = row['Category']
            sellingPrice = row['Selling Price']
            aboutProduct = row['About Product']
            productSpecification = row['Product Specification']
            image = row['Image']
            color = row['Color']
            productUrl = row['Product Url']

            print(row['Image'].split('|')[0])

            image = image.split('|')[0] if len(image) > 0 else ''

            categories = [c.strip() for c in category.split('|')] if isinstance(category, str) and len(category) > 0 else []

            imageVector = []
            if image != '':
                try:
                    img_resp = requests.get(image, stream=True)
                    img_emb = model.encode(Image.open(img_resp.raw), batch_size=128, convert_to_tensor=True, show_progress_bar=True)
                    imageVector = img_emb.tolist()
                except:
                    print('download image error: jump')
                    continue

            data = {
                'productName': productName.strip() if isinstance(productName, str) else '',
                'aboutProduct': aboutProduct.strip() if isinstance(aboutProduct, str) else '',
                'sellingPrice': sellingPrice.strip() if isinstance(sellingPrice, str) else '',
                'productSpecification': productSpecification if isinstance(productSpecification, str) else '',
                'categories': categories,
                'image': image,
                'imageVector': imageVector,
                'color': color if isinstance(color, str) else '',
                'productUrl': productUrl
            }

            yield {
                '_index': index_name,
                '_id': uuid.uuid4(),
                '_source': data
            }

    helpers.bulk(esclient, bulk_insert())


def main():
    ingest()
    # df = pd.read_csv('../amazon_products.csv')
    # df = df.reset_index()
    # print(df.iloc[2]['Color'])

if __name__ == '__main__':
    main()
```

Vector Database Part 14

Elasticsearch untuk solusi Hybrid Search

Melakukan Vector Search image to image search menggunakan Model sentence-transformers/clip-ViT-B-32 dan metrics Cosine Similarity

Menampilkan hasil pencarian dokumen dengan *query* gambar tertentu

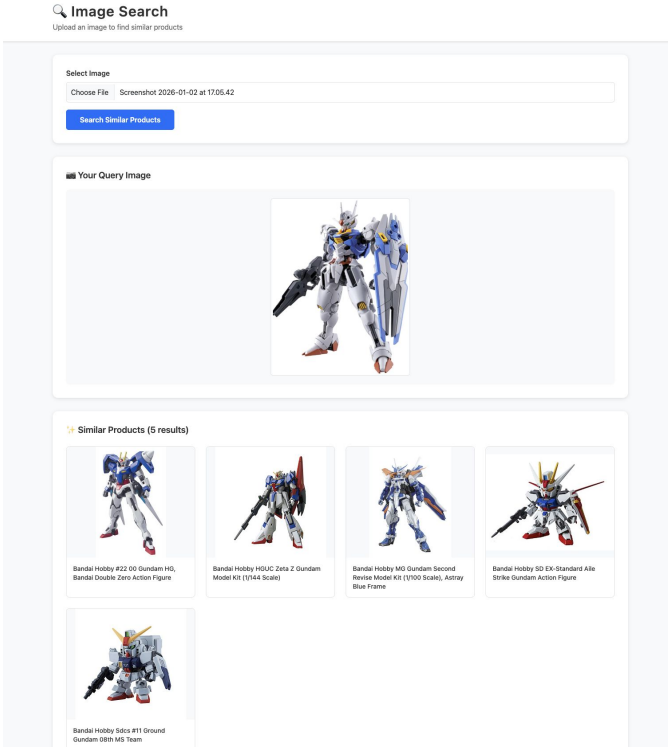
```python
# for elasticsearch 8.15.4
@router.post('/search/v2')
async def index(file: UploadFile):
    if not file:
        return {'success': False, 'message': 'file cannot be empty'}

    try:
        contents = await file.read()

        img_emb = model.encode(Image.open(io.BytesIO(contents)), batch_size=128, convert_to_tensor=True, show_progress_bar=True)

        search_payload = {
            "_source": False,
            "fields": [ "productName" , "image"],
            "knn": {
                "field": "imageVector",
                "query_vector": img_emb.tolist(),
                "k": 5,
                "num_candidates": 100
            }
        }

        response = esclient.search(
            index='products',
            body=search_payload
        )

        return {'success': True, 'message': 'search result', 'data': response}
    except Exception as e:
        log.error(e)
        return {'success': False, 'message': 'search error'}
    finally:
        await file.close()
```

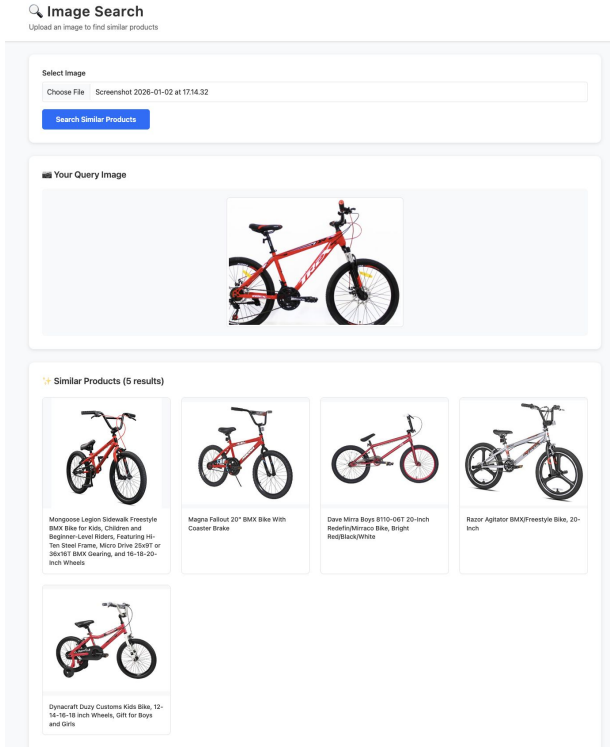
Vector Database Part 15

Elasticsearch untuk solusi Hybrid Search

Melakukan Vector Search image to image search menggunakan Model sentence-transformers/clip-ViT-B-32 dan metrics Cosine Similarity

Menampilkan hasil pencarian dokumen menggunakan *query* dengan gambar tertentu





Menggunakan *metrics* Cosine Similarity, **sentence-transformers/clip-ViT-B-32** sangat baik dalam memahami kesamaan visual dari dua gambar atau lebih **dengan cara membandingkan Vector Embedding dari gambar query dengan gambar yang ada di Index Elasticsearch**.

Vector Database Part 16

Honorable Mentions: Solusi Vector Store Lainnya

Selain tiga platform utama yang telah dibahas, terdapat beberapa solusi lain yang memiliki peran signifikan dalam industri:

1. Pinecone https://www.pinecone.io/

- **Karakteristik:** Vector Database berbasis *cloud-native* yang sepenuhnya dikelola (*fully managed*).
- **Keunggulan:** Menawarkan kemudahan operasional karena pengguna tidak perlu mengelola infrastruktur (*serverless*). Sangat populer di kalangan *Developer* karena integrasinya yang cepat dengan ekosistem **LLM** seperti OpenAI dan LangChain.

2. Milvus https://milvus.io/

- **Karakteristik:** *open-source*, skala Enterprise.
- **Keunggulan:** Menawarkan Vector Database performa tinggi dan *scalable*.

3. Chroma https://www.trychroma.com/

- **Karakteristik:** *open-source* dan dirancang khusus untuk memudahkan pembuatan aplikasi berbasis AI.
- **Keunggulan:** Menawarkan kemudahan. Fokus pada *developer experience* (DX) dengan kemudahan instalasi (bisa berjalan secara *in-memory* atau *on-premise*) dan sangat efisien untuk fase *prototyping* hingga produksi skala menengah.

Useful Links

- [Desmos] Vector Projection Dot Product: https://www.desmos.com/calculator/cd8w4ic6rt
- [Desmos] Dot Product https://www.desmos.com/calculator/617aaa4053
- [Desmos] Cosine Similarity https://www.desmos.com/calculator/678bd69c20
- [Desmos] Euclidean Distance https://www.desmos.com/calculator/bce9c211b0
- [Desmos] Simulate Euclidean Distance https://www.desmos.com/calculator/2qpnvqzjh2
- [Desmos] Trigonometry https://www.desmos.com/calculator/wgjxkrmeud
- [Desmos] Cross Product https://www.desmos.com/3d/961c13b698
- [Desmos] Vector Addition https://www.desmos.com/calculator/3a550e6b78
- [Desmos] Vector Multiplication with scalar https://www.desmos.com/calculator/745e2fd4c6
- [Desmos] 3D Vector https://www.desmos.com/3d/029e5691bc
- [Notebook Google Colab] Language Model for Embedding Model from scratch **https://colab.research.google.com/drive/1oTlaXgx9rTm3t0SxAuijMJnEZYk0_NB5?usp=sharing**
- [Notebook Google Colab] Using pre-trained Embedding Model **https://colab.research.google.com/drive/1uQIxkB5wJFzNmj2elg5lFxC7kBEE7Sma?usp=sharing**
- [Github] Image to Image search with Elasticsearch Vector Store and **sentence-transformers/clip-ViT-B-32** https://github.com/musobarlab/reactjs-elasticsearch-auto-complete-example/tree/master/imageembed

Tentang Penulis

Nama: **Wuriyanto**

Email: wuriyanto007@gmail.com

Website: https://wuriyan.to