Linear Algebra Introduction

Frank Burkholder (credit Taryn Heilman)

Objectives

- Be able to define scalars, vectors, and matrices
- Translate real-world data into scalars, vectors and arrays
    - Featurizing your data
- Describe operations that define vector length and distance between vectors
- Describe operations that multiply vectors and matrices
- Describe the python library NumPy and define its fundamental data structure
- Use NumPy to:
    - Create scalars, vectors, and matrices
    - Find the L1 and L2 norms of a vector
    - Find the euclidean, manhattan, and cosine distances between two vectors
    - Find the dot product of two matrices

Scalars

- A **scalar** is quantity that only has a magnitude (not a direction)
  - I did **2** loads of laundry.
  - The check was $**16.72**.
  - The answer is **-0.356**.
- Typographically represented as a lowercase variable:
  - ρ
  - t
  - x

Vectors

A **vector** is a quantity that has a magnitude and a direction. You can think of axes (columns) of data as directions that define the vector space of the data, and the values inside as the magnitude in each of those directions.

Bold, lowercase: **x**

An **array** is an ordered sequence of values.

A vector **is** an array.

Vectors

A **vector** is a quantity that has a magnitude and a direction. You can think of axes (columns) of data as directions that define the vector space of the data, and the values inside as the magnitude in each of those directions.

Bold, lowercase: **x**

Example:

- We were driving **30 mph** going **North** on **I25**.

Speed North	Speed East	On I25
30	0	True

Vectors

A **vector** is a quantity that has a magnitude and a direction. You can think of axes (columns) of data as directions that define the vector space of the data, and the values inside as the magnitude in each of those directions.

Bold, lowercase: **x**

Example:

- We were driving **30 mph** going **North** on **I25**.

Speed North	Speed East	On I25
30	0	True

$$\underbrace{\text{Speed North} \quad \text{Speed East}}_{\text{Numerical}} \quad \underbrace{\text{On I25}}_{\text{Categorical}}$$

Vectors

A **vector** is a quantity that has a magnitude and a direction. You can think of axes (columns) of data as directions that define the vector space of the data, and the values inside as the magnitude in each of those directions.

Bold, lowercase: **x**

Example:

- We were driving **30 mph** going **North** on **I25**.

Speed North	Speed East	On I25
30	0	1

Numerical

Categorical

Binary encoding
0 = False
1 = True
It's still categorical (but a numerical value).

Vectors - examples

- The well in **Colorado** was **8000 ft deep** and **10 ft wide** with a **100 gpm** capacity.

State	Depth	Width	Capacity
CO	8000	10	100

Vectors - examples

- The well in **Colorado** was **8000 ft deep** and **10 ft wide** with a **100 gpm** capacity.

State	Depth	Width	Capacity
CO	8000	10	100

What do these numbers mean?

(If you came back to this data two weeks from now, would you remember?)

Vectors - examples

- The well in **Colorado** was **8000 ft deep** and **10 ft wide** with a **100 gpm** capacity.

State_abrv	Depth_ft	Width_ft	Capacity_gpm
CO	8000	10	100

Consider including units of measure in column names.

Vectors - examples

- I bought **2 boxes of oatmeal**, **1 box of crackers**, at the **supermarket** for only **$2.50**.

Oatmeal	Crackers	Location	Price
2	1	supermarket	2.5

Vectors - examples

- I bought **2 boxes of oatmeal**, **1 box of crackers**, at the **supermarket** for only **$2.50**.

Oatmeal	Crackers	Location	Price
2	1	supermarket	2.5

$\uparrow$

It’s a categorical variable, but models can’t train on text. How to make this a number?

Vectors - examples

- I bought **2 boxes of oatmeal**, **1 box of crackers**, at the **supermarket** for only **$2.50**.

Oatmeal	Crackers	Location	Price
2	1	supermarket	2.5

↑

Let’s say options (categories) were:
supermarket, corner store, and on-line.
Use one-hot encoding.

Vectors - examples

- I bought **2 boxes of oatmeal**, **1 box of crackers**, at the **supermarket** for only **$2.50**.

Oatmeal	Crackers	supermarket	corner_store	on-line	Price
2	1	1	0	0	2.5

The purchase location has been one-hot encoded.

Vectors - examples

- I bought **2 boxes of oatmeal**, **1 box of crackers**, at the **supermarket** for only **$2.50**.

Oatmeal	Crackers	supermarket	Price
2	1	1	2.5

Then again, maybe it only matters if the purchased happened at the supermarket or not...

Vectors - examples

- Can you make a vector of this data-science haiku?

  My profile is me
  A bunch of features - that’s all
  But where is my soul?

Vectors - examples

- Can you make a vector of this data-science haiku?

  My profile is me
  A bunch of features - that’s all
  But where is my soul?

  Consider using a bag-of-words approach, where you count the number of times each word occurs (term frequency).

Vectors - examples

- Can you make a vector of this data-science haiku?

  My profile is me
  A bunch of features - that's all
  But where is my soul?

  Consider using a bag-of-words approach, where you count the number of times each word occurs (term frequency).

a	all	bunch	but	features	is	me	my	profile	soul	that's	where

Vectors - examples

- Can you make a vector of this data-science haiku?

    My profile is me
    A bunch of features - that's all
    But where is my soul?

  Consider using a bag-of-words approach, where you count the number of times each word occurs (term frequency).

a	all	bunch	but	features	is	me	my	profile	soul	that's	where

the vocabulary

Vectors - examples

- Can you make a vector of this data-science haiku?

  My profile is me
  A bunch of features - that's all
  But where is my soul?

  Consider using a bag-of-words approach, where you count the number of times each word occurs (term frequency).

a	all	bunch	but	features	is	me	my	profile	soul	that's	where
1	1	1	1	1	2	1	2	1	1	1	1

the vocabulary

the counts

Vectors - examples

- Can you make a 1-D vector (a row) describing the status of this tic-tac-toe game?

X		
	O	O
		X

Vectors - examples

- Can you make a 1-D vector (a row) describing the status of this tic-tac-toe game?

X		
	O	O
		X

Consider flattening this 2-D array into 1-D.
Values: -1 (unplayed), 0 (O), and 1 (X).

Vectors - examples

- Can you make a 1-D vector (a row) describing the status of this tic-tac-toe game?

1	-1	-1
-1	0	0
-1	-1	1

Top Left	Top Middle	Top Right
Middle Left	Middle	Middle Right
Bottom Left	Bottom Middle	Bottom Right

Consider flattening this 2-D array into 1-D.
Values: -1 (unplayed), 0 (O), and 1 (X).

Vectors - examples

- Can you make a 1-D vector (a row) describing the status of this tic-tac-toe game?

X		
	O	O
		X

1	-1	-1
-1	0	0
-1	-1	1

TL	TM	TR
ML	M	MR
BL	BM	BR

The 1-D vector:

TL	TM	TR	ML	M	MR	BL	BM	BR
1	-1	-1	-1	0	0	-1	-1	1

Vectors - moral of the story

- Almost any type of data can be vectorized/featurized (doesn’t always have to be 1-D).
- Almost always, there is more than one way to do it.
- Your ability to featurize and feature engineer your data will be fundamental to the success of your machine learning algorithms and your career as a data scientist.
- Your intuition will grow with experience (and by reading books, papers, and blogs!)

Vectors - breakout

Pair up. With your partner, come up with how you would vectorize (featurize) this data:

- The location and orientation of an aerial drone
- Stock price
- A black-and-white image
- A gray-scale image
- A color image
- Music you hear at a concert
- A movie review

Matrices

- A **matrix** is a rectangular array of numbers arranged in rows and columns. Can think of an array as rows of vectors, or columns of vectors. Bold, uppercase: **X**

Well	Depth	Width
1	8000	10
2	1000	3
3	10000	5

  If you like, a matrix is rows of observations, or samples.

- The term frequency matrix of these data-science haikus:

My profile is me
A bunch of features - that's all
But where is my soul?

As data unfold
Alas the model won't fit
Still the curve bends right

haiku	a	all	alas	as	bends	bunch	but	curve	data	featur es	is	model	my	profile	right	soul	still	that's	the	unfold	where	won't
1	1	1	0	0	0	1	1	0	0	1	2	0	2	1	0	1	0	1	0	0	1	0
2	0	0	1	1	1	0	0	1	1	0	0	1	0	0	1	0	1	0	2	1	0	1

Vector operations

- Length of a vector
    - L1 & L2 norms
- Distance between vectors
    - euclidean distance (L2)
    - manhattan distance (L1)
    - cosine distance
        - dot product
- Matrix multiplication
    - dot product

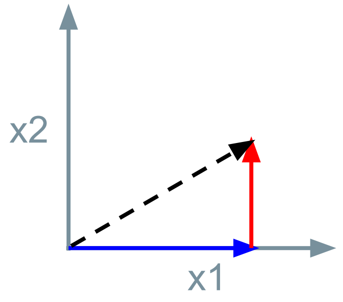
Length of a vector

Euclidean norm [edit]

Main article: Euclidean distance

On an *n*-dimensional Euclidean space **R**^{n}, the intuitive notion of length of the vector $\boldsymbol{x} = (x_1, x_2, ..., x_n)$ is captured by the formula

$$\|\boldsymbol{x}\|_2 := \sqrt{x_1^2 + \cdots + x_n^2}.$$



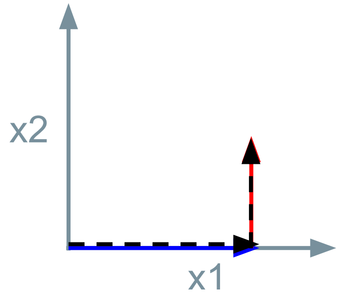
L2 norm: $\|\boldsymbol{x}\|_2 = \sqrt{x_1^2 + x_2^2}$

Taxicab norm or Manhattan norm [edit]

Main article: Taxicab geometry

$$\|\boldsymbol{x}\|_1 := \sum_{i=1}^{n} |x_i| .$$

The name relates to the distance a taxi has to drive in a rectangular street grid to get from the origin to the point *x*.



L1 norm: $\|\boldsymbol{x}\|_1 = |x_1| + |x_2|$

source: Wikipedia

Distance between vectors: Euclidean & Manhattan

The **Euclidean distance** between points **p** and **q** is the length of the line segment connecting them ($\overline{\mathbf{pq}}$).

In Cartesian coordinates, if **p** = (p_1, p_2,..., p_n) and **q** = (q_1, q_2,..., q_n) are two points in Euclidean *n*-space, then the distance (d) from **p** to **q**, or from **q** to **p** is given by the Pythagorean formula:[1]

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2 + (q_2-p_2)^2 + \cdots + (q_n-p_n)^2}$$

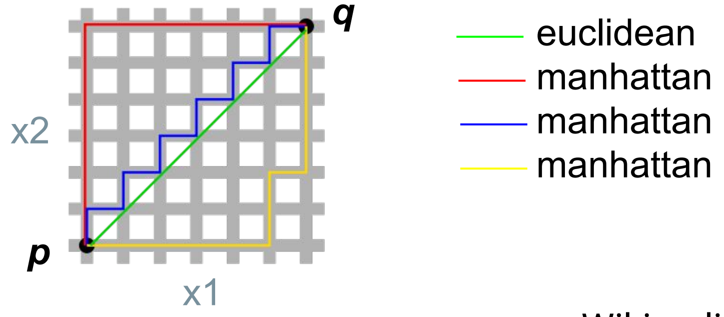
$$= \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}.$$

The taxicab distance, d_1, between two vectors $\mathbf{p}, \mathbf{q}$ in an *n*-dimensional real vector space with fixed Cartesian coordinate system, is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. More formally,

$$d_1(\mathbf{p},\mathbf{q}) = \|\mathbf{p}-\mathbf{q}\|_1 = \sum_{i=1}^{n}|p_i-q_i|,$$

where $(\mathbf{p},\mathbf{q})$ are vectors

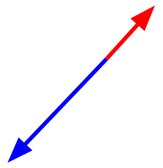
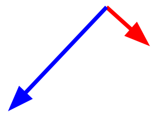
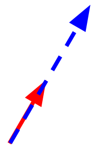
$$\mathbf{p} = (p_1, p_2, \ldots, p_n) \text{ and } \mathbf{q} = (q_1, q_2, \ldots, q_n)$$



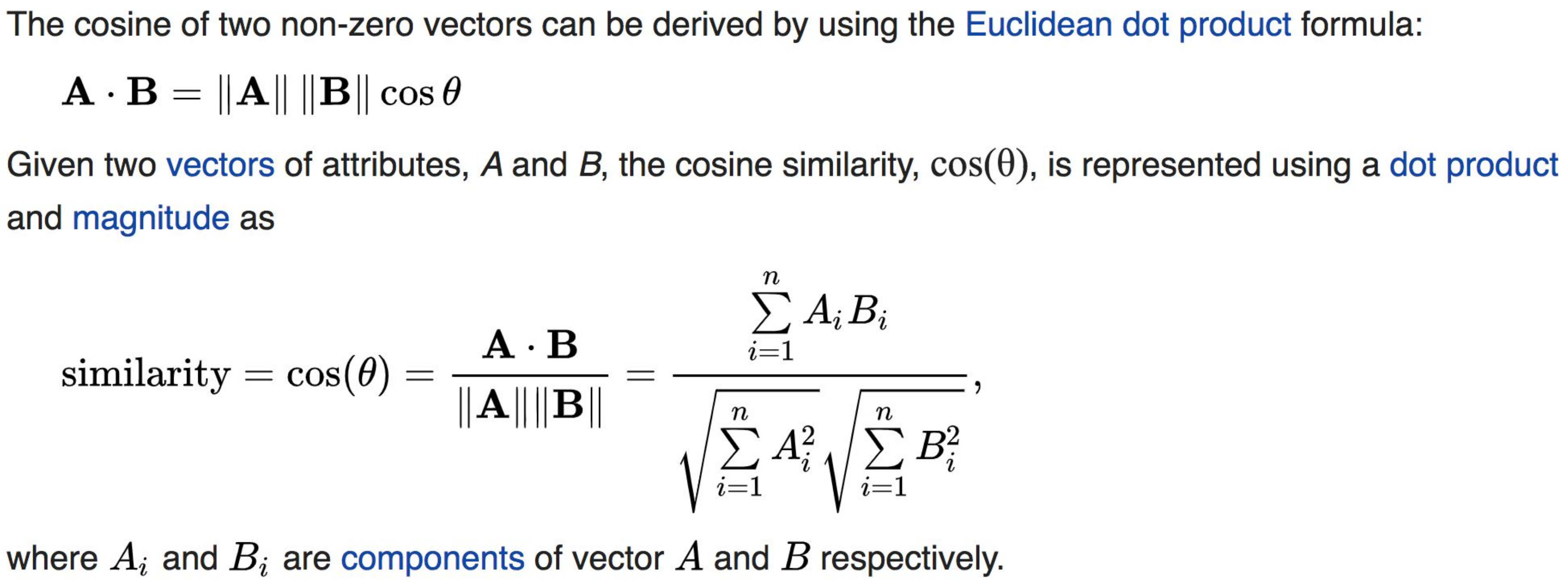
source: Wikipedia

Distance between vectors: Cosine

Cosine distance = 1 - cosine similarity
Are the vectors pointing in the same direction?
Typically used in higher dimensional vector spaces (easier to be similar)

Metric	Vectors pointing in opposite directions	Vectors orthogonal	Vectors pointing in the same direction
Image			
Cosine similarity	-1	0	1
Cosine distance	2	1	0

Calculation of Cosine Similarity



The cosine of two non-zero vectors can be derived by using the Euclidean dot product formula:

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$$

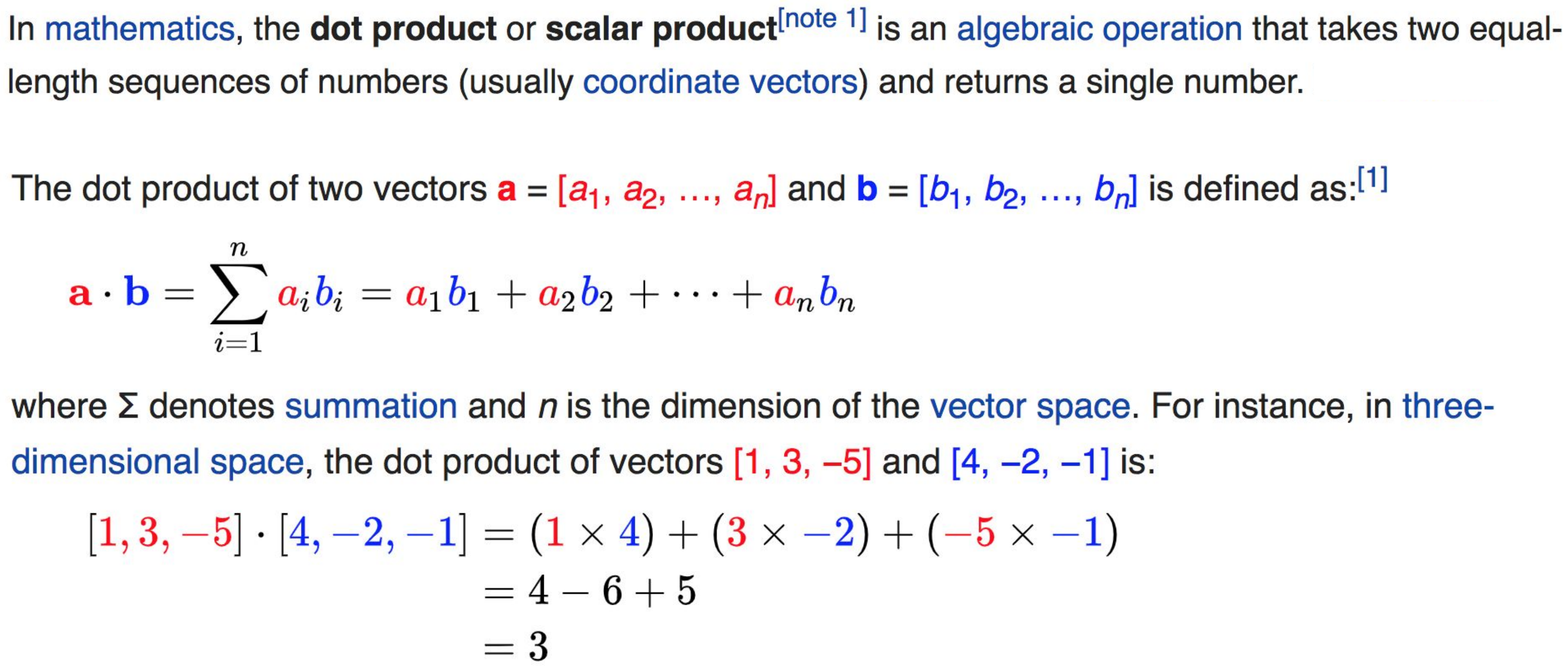
Given two vectors of attributes, *A* and *B*, the cosine similarity, $\cos(\theta)$, is represented using a dot product and magnitude as

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

where A_i and B_i are components of vector A and B respectively.

source: Wikipedia

Dot product



In mathematics, the **dot product** or **scalar product**[note 1] is an algebraic operation that takes two equal-length sequences of numbers (usually coordinate vectors) and returns a single number.

The dot product of two vectors **a** = [a_1, a_2, …, a_n] and **b** = [b_1, b_2, …, b_n] is defined as:[1]

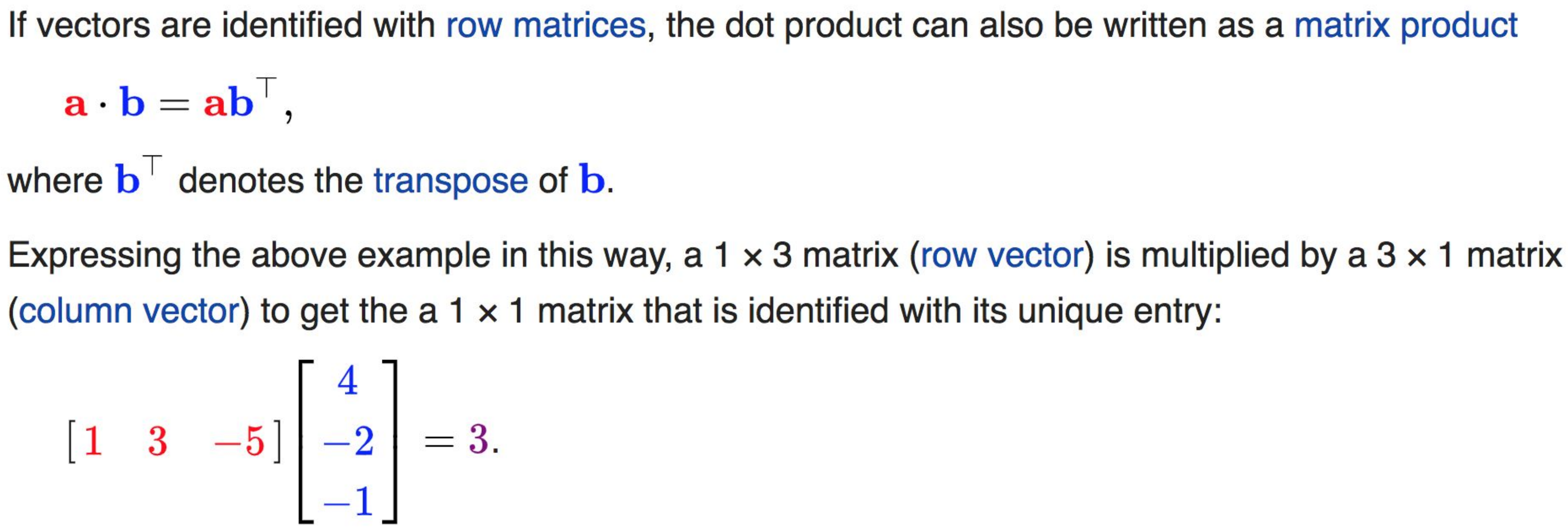
$$\mathbf{a}\cdot\mathbf{b}=\sum_{i=1}^{n}a_ib_i=a_1b_1+a_2b_2+\cdots+a_nb_n$$

where Σ denotes summation and *n* is the dimension of the vector space. For instance, in three-dimensional space, the dot product of vectors [1, 3, −5] and [4, −2, −1] is:

$$\begin{aligned}[1,3,-5]\cdot[4,-2,-1]&=(1\times 4)+(3\times -2)+(-5\times -1)\\&=4-6+5\\&=3\end{aligned}$$

source: Wikipedia

Dot product - extrapolating to matrix multiplication



If vectors are identified with row matrices, the dot product can also be written as a matrix product

$$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}\mathbf{b}^{\top},$$

where $\mathbf{b}^{\top}$ denotes the transpose of $\mathbf{b}$.

Expressing the above example in this way, a 1 × 3 matrix (row vector) is multiplied by a 3 × 1 matrix (column vector) to get the a 1 × 1 matrix that is identified with its unique entry:

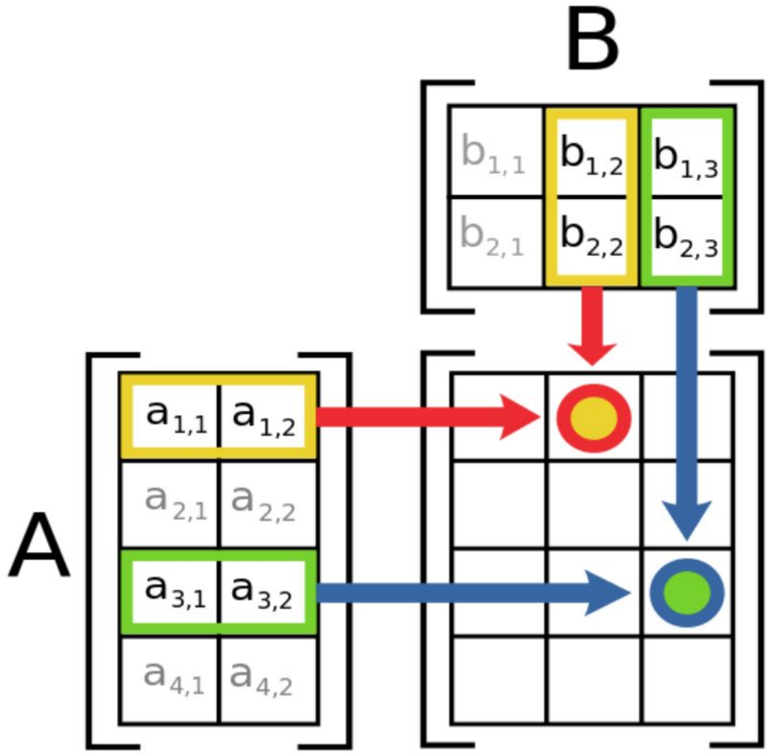
$$\begin{bmatrix} 1 & 3 & -5 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \\ -1 \end{bmatrix} = 3.$$

source: Wikipedia

Matrix Multiplication

The figure to the right illustrates diagrammatically the product of two matrices $\mathbf{A}$ and $\mathbf{B}$, showing how each intersection in the product matrix corresponds to a row of $\mathbf{A}$ and a column of $\mathbf{B}$.

$$\overset{4\times 2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \overset{2\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \overset{4\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & x_{12} & x_{13} \\ \cdot & \cdot & \cdot \\ \cdot & x_{32} & x_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$



source: Wikipedia

Breakout - pair up

A =

1	2
-1	0

B =

1	2	0
1	-1	3

1) What is A • B?
2) What is B • A?
3) Can you do B • B?
4) How about B • B^T?

NumPy

NumPy is the fundamental package for scientific computing in Python.

It provides:

- multidimensional array object, the ndarray
  - encapsulates *n*-dimensional arrays of homogeneous data types stored (ideally) sequentially in memory
- routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, **basic linear algebra**, basic statistical operations, random simulation and much more.
  - mostly written in C

There are a couple important differences between NumPy arrays and standard Python sequences:

- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an *ndarray* will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.

NumPy demo

`linear-algebra-with-numpy.ipynb`

Objectives

- Be able to define scalars, vectors, and matrices
- Translate real-world data into scalars, vectors and arrays
    - Featurizing your data
- Describe operations that define vector length and distance between vectors
- Describe operations that multiply vectors and matrices
- Describe the python library NumPy and define its fundamental data structure
- Use NumPy to:
    - Create scalars, vectors, and matrices
    - Find the L1 and L2 norms of a vector
    - Find the euclidean, manhattan, and cosine distances between two vectors
    - Find the dot product of two matrices