Linear Algebra

CENG 499
Introduction to Data Science

Erdoğan Doğdu

Content

- Vectors
- Matrices

Vectors

- Points in some finite-dimensional space
- Example:
  - heights, weights, and ages of people
  - three dimensional vector (python list)
    - height_weight_age = [70, *# inches,*
170, *# pounds,*
40] *# years*
  - 4 exam grades of students in a class
  - 4-dimensional vector
    - grades = [95, *# exam1*
80, *# exam2*
75, *# exam3*
62] *# exam4*

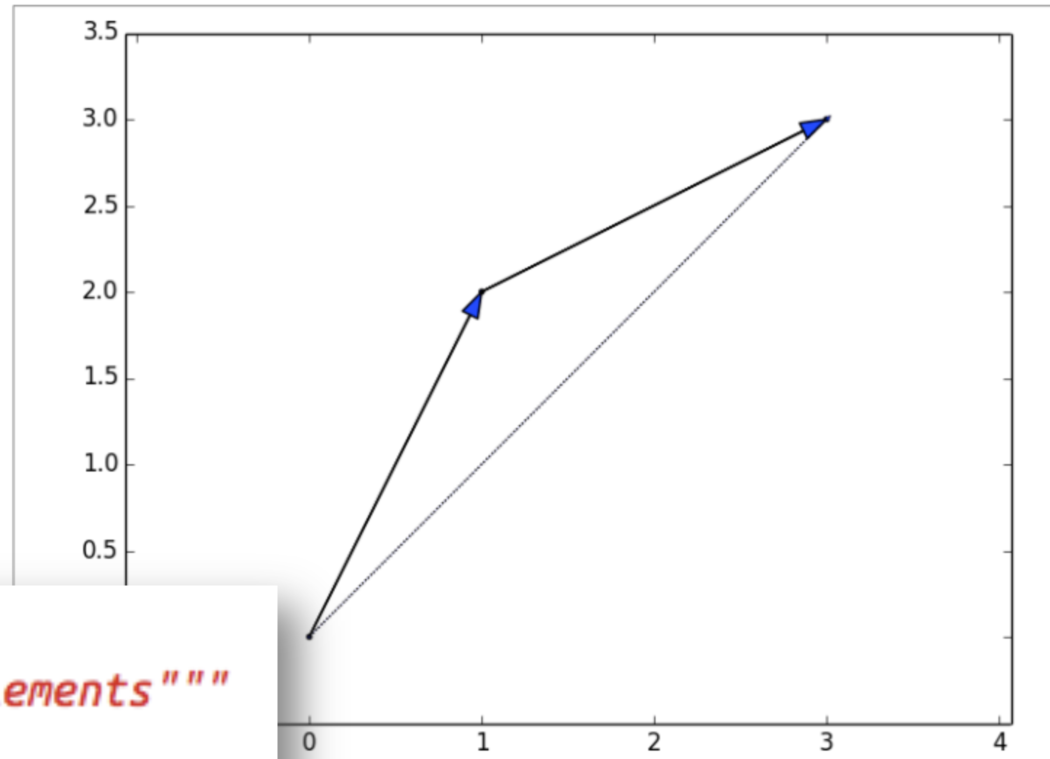
Vector operations

- Python lists are not vectors, so let's implement
- Add two vectors

  [1, 2] + [2, 1]

  = [1+2, 2+1]

  = [3, 3]



o vectors

```python
def vector_add(v, w):
    """adds corresponding elements"""
    return [v_i + w_i
            for v_i, w_i in zip(v, w)]
```

Vector operations

```python
def vector_subtract(v, w):
    """subtracts corresponding elements"""
    return [v_i - w_i
            for v_i, w_i in zip(v, w)]
```

```python
def vector_sum(vectors):
    """sums all corresponding elements"""
    result = vectors[0]                          # start with the first vector
    for vector in vectors[1:]:                   # then loop over the others
        result = vector_add(result, vector)      # and add them to the result
    return result
```

```python
def vector_sum(vectors):
    return reduce(vector_add, vectors)
```

Vector operations

```python
def scalar_multiply(c, v):
    """c is a number, v is a vector"""
    return [c * v_i for v_i in v]
```

```python
def vector_mean(vectors):
    """compute the vector whose ith element is the mean of the
    ith elements of the input vectors"""
    n = len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))
```

Vector operations

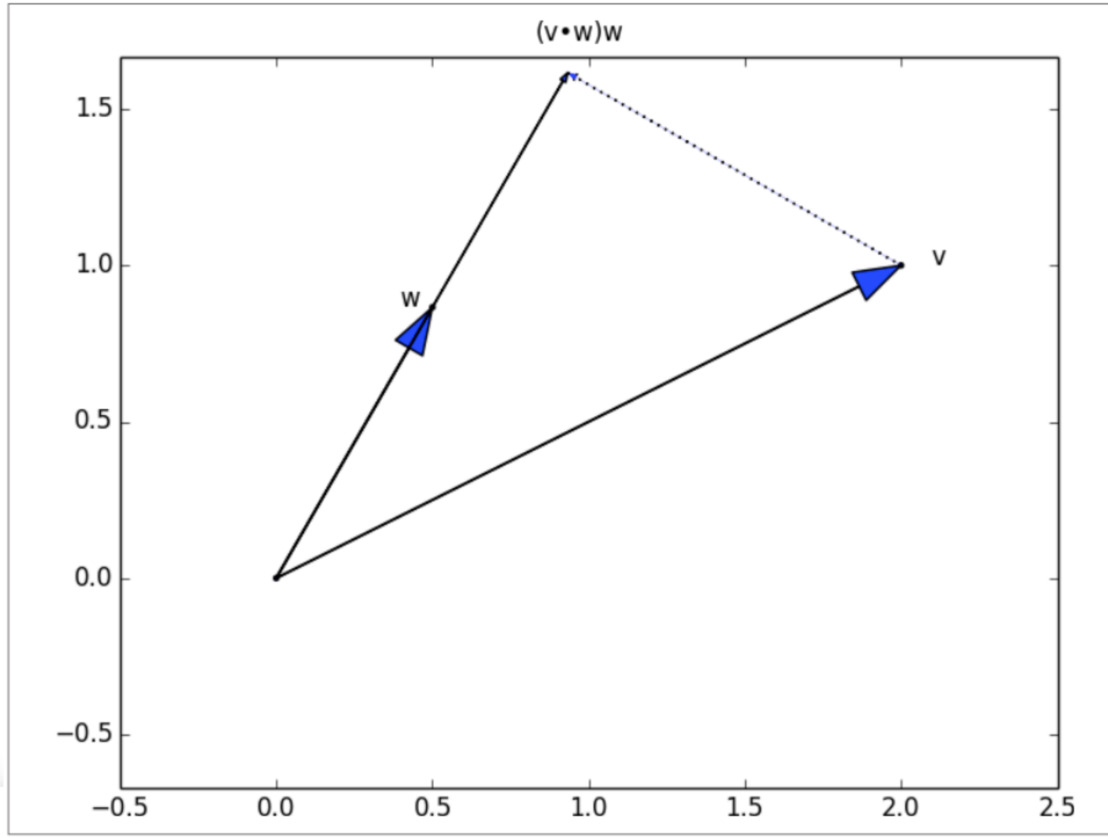


Figure 4-2. The dot product as vector projection

```
def dot(v, w):
    """v_1 * w_1 + ... + v_n * w_n"""
    return sum(v_i * w_i
               for v_i, w_i in zip(v, w))
```

Vector operations

```python
def sum_of_squares(v):
    """v_1 * v_1 + ... + v_n * v_n"""
    return dot(v, v)
```

```python
import math

def magnitude(v):
    return math.sqrt(sum_of_squares(v))   # math.sqrt is square root function
```

Vector operations

- Distance b/w two vectors

$$\sqrt{(v_1 - w_1)^2 + \ldots + (v_n - w_n)^2}$$

```
def squared_distance(v, w):
    """(v_1 - w_1) ** 2 + ... + (v_n - w_n) ** 2"""
    return sum_of_squares(vector_subtract(v, w))

def distance(v, w):
   return math.sqrt(squared_distance(v, w))
```

```
def distance(v, w):
    return magnitude(vector_subtract(v, w))
```

Vectors operations

- Using **lists as vectors** is great for exposition but **terrible for performance**.
- In production code, you would want to use the **NumPy** library, which includes a high-performance **array class** with all sorts of arithmetic operations included.

Matrices

- Two dimensional
- Lists of lists in python

```
A = [[1, 2, 3],  # A has 2 rows and 3 columns
     [4, 5, 6]]

B = [[1, 2],     # B has 3 rows and 2 columns
     [3, 4],
     [5, 6]]
```

Matrix operations

```
def shape(A):
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0   # number of elements in first row
    return num_rows, num_cols
```

```
def get_row(A, i):
    return A[i]             # A[i] is already the ith row

def get_column(A, j):
    return [A_i[j]          # jth element of row A_i
            for A_i in A]   # for each row A_i
```

Matrix ops

```
def make_matrix(num_rows, num_cols, entry_fn):
    """returns a num_rows x num_cols matrix
    whose (i,j)th entry is entry_fn(i, j)"""
    return [[entry_fn(i, j)             # given i, create a list
             for j in range(num_cols)]  #   [entry_fn(i, 0), ... ]
            for i in range(num_rows)]   # create one list for each i
```

```
def is_diagonal(i, j):
    """1's on the 'diagonal', 0's everywhere else"""
    return 1 if i == j else 0

identity_matrix = make_matrix(5, 5, is_diagonal)

# [[1, 0, 0, 0, 0],
#  [0, 1, 0, 0, 0],
#  [0, 0, 1, 0, 0],
#  [0, 0, 0, 1, 0],
#  [0, 0, 0, 0, 1]]
```

Matrix ops

- heights, weights, and ages of 1,000 people
  - 1, 000 × 3 matrix

```
data = [[70, 170, 40],
        [65, 120, 26],
        [77, 250, 19],
        # ....
       ]
```

Matrix ops

```
friendships = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
               (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

```
     #      user 0  1  2  3  4  5  6  7  8  9
     #
friendships = [[0, 1, 1, 0, 0, 0, 0, 0, 0, 0], # user 0
               [1, 0, 1, 1, 0, 0, 0, 0, 0, 0], # user 1
               [1, 1, 0, 1, 0, 0, 0, 0, 0, 0], # user 2
               [0, 1, 1, 0, 1, 0, 0, 0, 0, 0], # user 3
               [0, 0, 0, 1, 0, 1, 0, 0, 0, 0], # user 4
               [0, 0, 0, 0, 1, 0, 1, 1, 0, 0], # user 5
               [0, 0, 0, 0, 0, 1, 0, 0, 1, 0], # user 6
               [0, 0, 0, 0, 0, 1, 0, 0, 1, 0], # user 7
               [0, 0, 0, 0, 0, 0, 1, 1, 0, 1], # user 8
               [0, 0, 0, 0, 0, 0, 0, 0, 1, 0]] # user 9
```

Matrix ops

```
friendships[0][2] == 1    # True, 0 and 2 are friends
friendships[0][8] == 1    # False, 0 and 8 are not friends
```

```
friends_of_five = [i                                              # only need
                   for i, is_friend in enumerate(friendships[5])  # to look at
                   if is_friend]                                  # one row
```

Read more

- Linear Algebra, from UC Davis
  - https://www.math.ucdavis.edu/~linear/
- Linear Algebra, from Saint Michael's College
  - http://joshua.smcvt.edu/linearalgebra/