数据科学 2 线性代数

向量

向量指可以加总（以生成新的向量），可以乘以标量（即数字），也可以生成新的向量的对象。

向量是有限维空间的点，比如身高、体重、年龄数据可以表示为三维向量(height, weight, age)，考试成绩可以表示为四维向量 (exam1, exam2, exam3, exam4)

```
height_weight_age = [70, 170, 40]
grades = [95, 80, 75, 62]
```

向量加法

两个向量v和w长度相同，和等于一个新向量，新向量第一个元素等于v[0] + w[0], 第二个元素等于v[1] + w[1] (向量长度不同，不能相加)
向量 [1, 2] + [2, 1] = [1 + 2, 2 + 1] = [3, 3]

```
def vector_add(v, w):
    """adds two vectors componentwise"""
    return [v_i + w_i for v_i, w_i in zip(v,w)]
```

向量减法

```
def vector_subtract(v, w):
    """subtracts two vectors componentwise"""
    return [v_i - w_i for v_i, w_i in zip(v,w)]
```

多向量累加

```
# 方法1
def vector_sum(vectors):
    result = vectors[0]
    for vector in vectors[1:]:
```

```
        result = vector_add(result, vector)
    return result
```

```
# 方法2
from functools import partial, reduce
def vector_sum(vectors):
    return reduce(vector_add, vectors)
```

向量乘以标量 = 向量每个元素乘标量

```
def scalar_multiply(c, v):
    return [c * v_i for v_i in v]
```

一堆向量（长度相同）的均值

```
def vector_mean(vectors):
    n = len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))
```

向量点乘

两个向量点乘表示对应元素的分量乘积之和

```
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))
```

向量平方和

```
def sum_of_squares(v):
    return dot(v, v)
```

计算向量的大小（或长度）

```
import re, math, random # regexes, math functions, random numbers

def magnitude(v):
    return math.sqrt(sum_of_squares(v))
```

两个向量的距离

```
def squared_distance(v, w):
    return sum_of_squares(vector_subtract(v, w))

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

矩阵

如果A是一个矩阵，那么A[i][j]就表示第i行第j列的元素

```
# 2行3列
A = [[1, 2, 3],
     [4, 5, 6]]

# 3行2列
B = [[1, 2],
     [3, 4],
     [5, 6]]
```

矩阵A有len(A)行和len(A[0])列，叫形状(shape)

```
def shape(A):
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0
    return num_rows, num_cols
```

如一个矩阵有n行k列，则表示为n x k 矩阵。可以把这个矩阵每一行都当作长度为k的向量，每

一列当作长度为n的向量

```
def get_row(A, i):
    return A[i]

def get_column(A, j):
    return [A_i[j] for A_i in A]
```

根据形状和生成元素的函数创建矩阵

```
def make_matrix(num_rows, num_cols, entry_fn):
    return [[entry_fn(i, j) for j in range(num_cols)]
            for i in range(num_rows)]
```

生成对角线元素1，其他元素为0函数

```
def is_diagonal(i, j):
    return 1 if i == j else 0
```

矩阵用途：

- 每个向量看作矩阵的一行，用矩阵表示一个包含多维向量的数据集
- 表示二维关系，之前将关系表示为数据对(i, j)，还可以通过矩阵来表示。
如果节点i和节点j有关系，用矩阵A[i][j]为1来表示

```
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
```

缺点：关系少的时候，内存利用效率低

优点：确认节点是否连接很快

```
friendships[0][2] == 1
friendships[0][8] == 1
```

查找一个节点的所有连接

```
friends_of_five = [i for i, is_friend in enumerate(friendships[5]) if
is_friend]
```