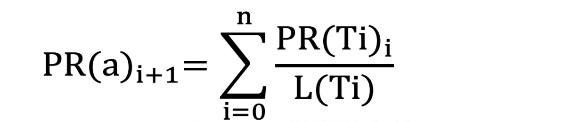
PageRank

背景

PageRank算法最初作为计算互联网网页重要度的算法被Google提出的，基于网页链接分析对关键词匹配搜索结果进行处理。它是定义在网页集合上的一个函数，它对每个网页给出一个正实数值，表示网页的重要程度，PageRank值越高，表示该网页越重要。如果一个网页被很多网页链接则其PageRank值会相对较高，另一方面一个PageRank值很高的网页链接到其它的网页，那么被链接的网页的PageRank值也会因此而提高。由于网络中网页链接的相互指向，该分值的计算为一个迭代过程，最终网页根据所得分值进行检索排序。实际应用中网页的数据主要以图的形式存在，PageRank是找出图中顶点（网页链接）的重要性。

PR（a）表示当前节点a的PR值

PR（Ti）表示其他各个节点（能够指向a）的PR值

L（Ti）表示其他各个节点（能够指向a）的出链数

i代表当前时刻或迭代次数

Google给每一个网页都赋予一个初始,然后根据PageRank算法计算其。事实上，根据以上公式进行的PR计算并不是仅仅一次就结束的，由于网页之间的相互链接，任一网页的变化，都会引起其他与之有链接关系的网页的的变化，因此，确定某网页的，需要进行多次重复的计算。而在经过一定次数的重复计算之后，各网页的基本上达到稳定。PageRank算法使所有网页的总和非常巧妙地达到了平衡，所有网页的总和在每一次计算后也是保持不变的。假设Google的索引页有10亿个，每页的初始都是1，这些网页的总和则是10亿。在每次计算之后，无论个体网页的怎么变化，但是这些网页的总和始终保持10亿。

矩阵优化

随机游走的特点是一个结点到有向边连出的所有结点的转移概率相等，设一共有n个网页，则可以组织成一个n维矩阵，矩阵的行id表示有向边的目的节点id，列id表示有向边的起始节点id。其中第i行j列的值表示用户从页面j转到页面i的概率，这样的一个矩阵叫做转移矩阵（Transition Matrix）。转移矩阵表示为：

的取值根据页面的链接情况。如果结点有个有向边连出，并且结点是其连出的一个结点，则 ，否则。

随机游走在某个时刻访问各个结点的概率分布就是马尔可夫链在时刻的状态分布，可以用一个维**列向量**表示，那么在时刻访问各个结点的概率分布满足

笔记：要算某一刻到达节点i的概率，先算之前的时刻到达节点i的入边邻居的概率，然后累积上各入边邻居传递到节点i的概率

由此，在时刻访问各个结点的概率分布为

通常取值为向量。

若极限

存在，则极限向量满足马尔科夫链的平稳分布，即

平稳分布称为有向图的PageRank。的各个分量为各个结点的PageRank值，即

其中，，表示页面的PageRank值。

PageRank算法的迭代公式如下：

笔记1：d是阻尼系数，公式后半部分是，以便于加入到Rk上

笔记2：为了处理那些“没有向外链接的页面”（这些页面就像“黑洞”会吞噬掉用户继续向下浏览的概率）带来的问题，d=0.85（这里的d被称为阻尼系数（damping factor），其意义是，在任意时刻，用户到达某页面后并继续向后浏览的概率。1-d=0.15就是用户停止点击，随机跳到新URL的概率）的算法被用到了所有页面上，估算页面可能被上网者放入书签的概率。

表示第轮迭代中产生的PageRank向量，表示阻尼系数，满足,是有向图的转移矩阵，表示有向图顶点数量。

下列算法展示了PageRank的迭代算法。根据迭代公式计算每一轮的PageRank向量。如果迭代轮数可以不超过预先规定的上限提前结束，则可以得到PageRank向量，根据向量的分量得出各页面的重要度排名。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 算法伪代码：PageRank () | | | | | | | |
| 输入：有向图，初始向量，阻尼因子，迭代的上限 | | | | | | | |
| 输出：PageRank向量 | | | | | | | |
|  | | | 根据边集产生转移矩阵 | | | | |
|  | | | 初始化迭代次数; | | | | |
|  | | | while() do | | | | |
|  | | | | 计算 | | | |
|  | | | | if(与充分接近) then | | | |
|  | | | ; | | | |
|  | | | return ; | | | |
|  | | | end if | | | |
|  | | | | ; | | | |
|  | | | | end while | | | |
|  | | | return ; | | | | |

算法伪代码 PageRank算法

源码

import time

import numpy as np

import mindspore.ops as ops

from mindspore import dtype as mstype

from mindspore import Tensor

def read\_edges(filename):

"""

从边文件中读取边数据，并返回边列表和顶点集合。

Args:

filename (str): 边文件名。

Returns:

tuple: 包含边列表和顶点集合的元组。

"""

edges = []

vertices = set()

with open(filename, 'r') as file:

for line in file:

start, end = map(int, line.strip().split())

edges.append((start, end))

vertices.add(start)

vertices.add(end)

return edges, vertices

def build\_adjacency\_matrix(edges, vertices):

"""

根据边列表和顶点集合构建邻接矩阵，并返回邻接矩阵和顶点ID映射。

Args:

edges (list): 边列表。

vertices (set): 顶点集合。

Returns:

tuple: 包含邻接矩阵和顶点ID映射的元组。

"""

id\_map = {v: i for i, v in enumerate(sorted(vertices))}

n = len(id\_map)

matrix = np.zeros((n, n), dtype=np.float32) # 使用浮点数类型

for edge in edges:

start, end = edge

matrix[id\_map[start], id\_map[end]] = 1

return matrix, id\_map

def calculate\_pagerank(adjacency\_matrix, damping\_factor=0.85, epsilon=1e-6):

"""

计算PageRank值。

Args:

adjacency\_matrix (numpy.ndarray): 邻接矩阵。

damping\_factor (float, optional): 阻尼因子，默认为0.85。

epsilon (float, optional): 收敛阈值，默认为1e-6。

Returns:

numpy.ndarray: 包含每个节点的PageRank值的数组。

"""

n = adjacency\_matrix.shape[0]

deg\_out = np.sum(adjacency\_matrix, axis=1)

transfer\_matrix = adjacency\_matrix / np.where(deg\_out[:, np.newaxis] != 0, deg\_out[:, np.newaxis], 1)

pagerank = np.ones(n, dtype=np.float32) / n

pagerank = Tensor(pagerank, mstype.float32) # 使用浮点数类型的张量

transfer\_matrix = Tensor(transfer\_matrix, mstype.float32) # 使用浮点数类型的张量

start\_time = time.time() # 记录计算时间

while True:

new\_pagerank = (1 - damping\_factor) / n + damping\_factor \* ops.matmul(transfer\_matrix, pagerank)

norm\_diff = ops.norm(new\_pagerank - pagerank, 0) # 计算差异的L0范数

if norm\_diff < epsilon:

break

pagerank = new\_pagerank

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"PageRank计算时间: {elapsed\_time}秒")

return pagerank

def write\_pagerank(filename, pagerank, id\_map):

"""

将PageRank结果写入文件。

Args:

filename (str): 结果文件名。

pagerank (numpy.ndarray): 包含每个节点的PageRank值的数组。

id\_map (dict): 顶点ID映射。

Returns:

None

"""

with open(filename, 'w') as file:

for v, pr in id\_map.items():

file.write(f"{v} {pagerank[pr]}\n")

if \_\_name\_\_ == "\_\_main\_\_":

# 输入文件名

edge\_filename = "E:\华科实验室论文\MyPythonCode\pagerank\Wiki-Vote.txt" # 原始数据集文件

result\_filename = "E:\华科实验室论文\MyPythonCode\pagerank\pagerank8.txt" # 结果文件

# 读取边数据和顶点集合

start\_time = time.time()

edges, vertices = read\_edges(edge\_filename)

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"读取边数据时间: {elapsed\_time}秒")

# 构建邻接矩阵和顶点ID映射

start\_time = time.time()

adjacency\_matrix, id\_map = build\_adjacency\_matrix(edges, vertices)

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"构建邻接矩阵时间: {elapsed\_time}秒")

# 计算PageRank

pagerank = calculate\_pagerank(adjacency\_matrix)

# 写入结果文件

start\_time = time.time()

write\_pagerank(result\_filename, pagerank, id\_map)

end\_time = time.time()

elapsed\_time = end\_time - start\_time

print(f"写入结果文件时间: {elapsed\_time}秒")

print("PageRank计算完成，结果已写入pagerank8.txt文件。")