# COMP3074人- ai交互实验2:信息检索、表示和相似性

J´er´emie秘密地

2023年10月

## 1 介绍

在这个实验室里，你将构建一个搜索引擎。它会非常低效，当然也不会与谷歌竞争，但它会让你洞察任何时候你在使用基于搜索的算法时发生的基本操作:搜索引擎、推荐系统(亚马逊、Spotify、Netflix)等。

## 2 交易的工具

### 2.1 确保Numpy和Scipy已经安装

两个基本的Python库通常用于实现线性代数和科学计算函数:Numpy(用于数值方法和线性代数)和Scipy(用于科学计算)。如果你已经安装了Anaconda，它们已经在你的系统上了，你只需要像这样导入它们:

```
进口numpy
进口scipy
```

numpy的优点之一是它对矩阵的操作非常快，你可以通过它的数组数据结构访问矩阵。

### 2.2 酸洗对象

Python有一组方便的函数用于保存和加载对象，称为pickle，以及通过joblib进行操作的新方法。两者都能工作，但joblib对于大型机器对象来说稍好一些。在这里，我们把任何类型的实例化数据结构都称为对象。它可以是语言模型，机器学习模型，或者你喜欢的任何东西。

黄瓜:

```
import pickle
with open("filename.pickle", "wb") as f:
    pickle.dump(someobject, f)
with open("filename.pickle", "rb") as f:
    someobject = pickle.load(f)
```

With **joblib**:

```
from joblib import dump, load
dump(someobject, 'filename.joblib')
loaded_object = load('filename.joblib')
```

如果你正在构建一个使用复杂结构(如分类器或索引)的应用程序，这很重要，因为你不想重新计算它。这段代码并不意味着按原样运行;它是作为两个可以帮助你工作的函数的示例提供的。

# 3 术语文档矩阵

## 3.1 建立索引

搜索引擎的基本数据结构是词-文档矩阵。词项-文档矩阵仅仅是一个文档-词项矩阵的转置:我们关心的不是哪些词项出现在一个文档中，而是哪些文档包含一个特定的词项。也就是说，它只在处理某些搜索索引优化时才有意义，并且Term-Document和Document-Term矩阵在当前形式中基本上是等效的，所以如果它有助于您更好地理解它，请随意在您的头脑中翻转轴。我只是使用Term-Document而不是Document-Term，因为它是一种信息检索用法。TD矩阵的一个轴是词汇表(我们正在建模的术语)，而另一个轴是文档集合。

词汇表在表1中表示，相应的词-文档矩阵在表2中表示。

| TermId | Term |
|---|---|
| t1 | Term1 |
| t2 | Term2 |
| t3 | Term3 |
| t4 | Term4 |

表1:词汇表

| Term | Doc1 | Doc2 | Doc3 | Doc4 |
|---|---|---|---|---|
| **t1** | frequency(t1,d1) | frequency(t1,d2) | frequency(t1,d3) | frequency(t1,d4) |
| **t2** | frequency(t2,d1) | frequency(t2,d2) | frequency(t2,d3) | frequency(t2,d4) |
| **t3** | frequency(t3,d1) | frequency(t3,d2) | frequency(t3,d3) | frequency(t3,d4) |
| **t4** | frequency(t4,d1) | frequency(t4,d2) | frequency(t4,d3) | frequency(t4,d4) |

表2:Term-Document矩阵

在现实生活中，将这样的东西存储在一个固定大小的矩阵中会非常浪费，因为它的稀疏性(大量的0)，并且为稀疏矩阵建立了特定的数据结构。但是为了学习，我们可以处理一点资源的浪费。

词汇表的构建过程与我们在文本分类中看到的步骤相同:单词需要分词，然后基于首选的方法(大小写折叠、词干化/词干提取)进行标准化。更类似的是，停用词和低/高频词也可以被删除。一个出现在99%文档中的单词对于搜索来说是无用的，因为搜索它应该返回所有文档。

最后，每个词在每个文档中的频率通常也会以与我们在上一个实验中看到的非常相似的方式进行加权。TF-IDF的词项权重方案实际上来自于信息检索文献。

### 3.2 查询

就搜索引擎而言，查询只是另一个文档。因此，当用户输入一个搜索查询时，它只是简单地映射到从文档集合的词汇表中归纳出来的向量空间上。

# 4 相关性和相似性

信息检索系统的关键功能是计算文档与查询的相关性。因为查询被简单地认为是一个新文档，一个文档相对于一个文档的相关性可以简单地通过相似性度量来近似。当然，在真实的搜索引擎中，有更多的作用，但相似性对于我们当前的用例来说已经绰绰有余。

| Term | Doc1 | Doc2 | Doc3 | ... | Query document |
|---|---|---|---|---|---|
| **t1** | frequency(t1,d1) | frequency(t1,d2) | frequency(t1,d3) | ... | frequency(t1,q) |
| **t2** | frequency(t2,d1) | frequency(t2,d2) | frequency(t2,d3) | ... | frequency(t2,q) |
| **t3** | frequency(t3,d1) | frequency(t3,d2) | frequency(t3,d3) | ... | frequency(t3,q) |
| **t4** | frequency(t4,d1) | frequency(t4,d2) | frequency(t4,d3) | ... | frequency(t4,q) |

表3:带查询的词-文档矩阵

因此，搜索引擎的目标是通过降低相关性对所有文档进行排序，其中相关性将被定义为查询文档和每个文档之间的余弦相似度。

### 4.1 余弦相似度

查询文档与文档之间的余弦相似度d是由词汇表诱导的向量空间中文档向量与查询向量之间夹角的余弦，定义如下:

$$\text{similarity}(q,d) = \frac{q \cdot d}{\|q\| \cdot \|d\|} = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2}\sqrt{\sum_{i=1}^{n} d_i^2}} \tag{1}$$

在公式1中，$q_i$和$d_i$对应于查询和文档向量的$i^{th}$分量。例如，$q_2$对应的是查询文档中词条2的频率。N对应向量中分量的个数，即词汇量的大小。查询中不在词汇表中的词条会被忽略。

你可以自由地手动实现余弦相似度，因为它不是很复杂，但一些库使它变得容易得多。例如，Scipy实现了余弦距离。

```
from scipy import spatial

query_doc =  [1, 2, 0, 0]
document_1 = [2, 4, 0, 0]
sim_1 = 1 - spatial.distance.cosine(query_doc, document_1) # takes the inverse of
     the distance in order to get similarity
```

另一方面，Numpy为你提供了更容易实现它的工具。

```
from numpy import dot
from numpy.linalg import norm

query_doc =  [1, 2, 0, 0]
document_1 = [1, 5, 0, 0]

sim_1 = dot(query_doc, document_1)/(norm(query_doc)*norm(document_1))
```

注意，dot是你可以在公式(1)的中间部分看到的点积:q·d，而norm代表$l_2$-norm，符号为∥x∥。

# 5 搜索

所以现在你有了相关函数(余弦相似度)，索引(词项/文档或文档/词项矩阵)，你就可以真正构建一个搜索引擎了!这看起来可能让人望而生畏，那么让我们来回顾一下其中涉及到的组件:

## 5.1 通用算法

下面是建立索引时应该遵循的通用算法:

```
Compute the vocabulary of the corpus
Use the vocabulary to compute the term-document matrix of the corpus
Apply term weighting technique to that term-document matrix
```

And the general algorithm for search:

```
Apply tokenising, stemming, and other normalisation techniques on search query
Map search query on the document collection-induced vector space
Apply term weightign technique to the tokens in the search query
For each document in the document collection:
  Compute the cosine similarity with the search query
Rank documents by decreasing similarity
```

## 5.2 解析文档列表以构建索引

在实验1中我们看到了如何读取文件。

```
with open('document.txt', 'r', encoding='utf-8') as f:
    for line in f:
        print(line)  # print the current line
```

我还提到了使用os库(Python标准库的一部分)来解析一个装满文档的文件夹。

```
import os

document_path = "data"  # We assume the documents are stored in a folder named
    data, positioned in the folder where your Python code is

corpus = {}
for file in os.listdir(document_path):
    filepath = document_path + os.sep + file
    with open(filepath, encoding='utf8', errors='ignore', mode='r') as document:
        content = document.read()
        document_id = file
        corpus[document_id] = content
```

现在，我们的文档都存储在Python字典和语料库中，并准备被分析。当然，用这种方式来构建一个大规模的搜索引擎是很糟糕的，但这并不是我们现在真正在做的事情。

至于分词(tokenising)、案例折叠(case folding)和通用文本标准化(general text normalization)，你可以自由地推出自己的实现(比看起来要简单)，或者重用我们在实验室1中看到的一些实现，使用NLTK。

注意:你可以使用pickle和joblib(参见2.2节)来保存和加载索引，这样你就不必在每次运行程序时重新计算它。

## 5.3 要求用户进行查询

除了构建基于web的GUI, Python的input()函数是要求用户直接输入搜索查询的最简单方法。下面的代码片段不断请求新的查询，并将其存储在query变量中，直到用户输入停止。

```
stop = False
while not stop:
    query = input("Enter your query, or STOP to quit, and press return: ")

    if query == "STOP":
        stop = True
    else:
        print(f'You are searching for {query}')
        # your query processing comes here
```

## 5.4 置信度阈值和备用机制

有一点我没有涉及，但很重要的是备用机制的概念。当我们使用相似性搜索结果时，总是有可能查询不匹配我们语料库中的任何内容。这通常是通过我们相似度中的某种阈值来计算的，在这个阈值下，我们认为没有任何东西是相关的。然后我们该怎么做?我们有几个可能的选择。

### 5.4.1 通过警告消息回退

从这样的状态中恢复的最简单的方法就是简单地向用户显示没有任何与查询相关的内容，并建议他们重新表述或添加额外的词条。这与完全不显示结果不同，这可能会导致用户认为系统不起作用。

### 5.4.2 通过查询重新制定回退

再往前走一步，就是建议自己进行查询重构。一个简单的方法是使用语言模型(例如，给定一个现有的查询q，用户输入的下一个词条最有可能是什么?)

# 6 评估你的搜索结果

这就留给我们一个问题:你怎么知道你的搜索算法好不好?在信息检索中有几个非常有用的指标，可以让你了解你的搜索引擎是否成功。我们将看到其中3个:精确率，召回率，归一化折扣累积增益(NDCG)。但首先，我们需要先给自己一个黄金标准来衡量算法的优劣。

## 6.1 人工相关性评估

通常，我们根据一组我们知道预期结果的参考查询来衡量搜索引擎的质量。这些结果可以是二元的，即一个文档与一个查询相关或不相关，或者是分级的，即一个文档非常相关，有点相关，或不相关。在后一种情况下，相关性被编码为从0到2的数字，而在前一种情况下，它被编码为二进制数字0或1。

例如，一个示例查询将表示为$q_1$= {$d_1$:2,$d_5$:2,$d_6$:2,$d_7$:1,$d_8$:1}，隐式假设它对于所有其他文档都是0。知道了这一点，你就可以计算几个指标。

## 6.2 精度

精度@ K (Precision at K)是指返回的K个文档中有多少是相关的。通常是用多个K值绘制精度在K，以观察算法在需要返回更多数据时如何变化。而且，它有一个直观的意义:如果你只能在搜索引擎的第一个结果页面上显示K个文档，你不希望它被无关文档污染。精确度通常是用二元相关性判断来评估的，这意味着一个文档要么被认为是相关的，要么被认为是不相关的，对它的相关性没有分级。

$$\text{Precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

编写自己的Precision分数相对容易，但如果您正在寻找参考实现，Scikit-Learn有一个[1]。

## 6.3 回忆

Recall @ K (Recall at K)与检索了多少相关文档有关。和精度一样，通常用多个K值绘制召回率@ K，以查看随着返回更多结果，算法的结果如何变化。召回率在小型、专门的搜索引擎中更有意义，在这些引擎中，只有一组受限的文档真正被认为是相关的:医疗信息检索、法律信息检索等。它与精确率结合使用，可以全面展现搜索引擎的能力。

$$\text{Recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

编写自己的回忆分数相对容易，但如果你正在寻找一个参考实现，Scikit-Learn有一个[2]。

## 6.4 归一化贴现累积增益

归一化贴现累积增益(NDCG) @ p是一种以更细粒度的方式评估搜索结果的方法，不仅考虑是否返回相关文档，还考虑它们在结果集中的顺序。正如您在下面的DCG等式中看到的，$log_2 i$逐渐降低了$rel_i$的重要性，因此返回文档列表开头的结果被判断为比返回文档列表底部的结果更重要(它们被一个较小的分母除以)(它们被一个较大的分母除以)。然后将DCG除以理想DCG，理想DCG是完美排名的DCG值(所有高度相关的文档按相关性降序排列在列表的开头)。

$$DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2 i + 1}$$

$$NDGC_p = \frac{DCG_p}{iDCG_p}$$

其中IDCG是基于完美排序的DCG，即按用户定义相关性递减排序的文档的DCG。与其他指标一样，构建自己的NDCG计算器并不困难，而且是一种很好的学习方法，但是Scikit-Learn提供了一个参考实现[3]。

1 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
2 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
3 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ndcg_score.html

## 6.5 评估活动

评估活动是大型的定期活动，在此期间，搜索引擎和研究小组竞相使用巨大的数据集构建最佳的搜索算法。你可能想知道，为了评估那些搜索引擎，谁有时间浏览十亿多页，并为它们标注与给定查询相关或不相关的内容。事实是，信息检索界使用了一个聪明的技巧来避免这种标记:组合。池化的过程，每次搜索引擎评估活动都会发生:

1. 竞争的搜索引擎运行一组评估查询，并记录他们的搜索引擎的前N个结果

2. 中央机构恢复这些结果，并获取搜索期间返回的所有文档的并集

3.然后，该文档组合被随机分割，以便多个竞争对手必须对它们进行标记

4. 参赛者会收到一个测试结果的子集来标注

5. 然后，基于来自竞争对手的聚合标签，文档被分配了相关性

通过这种方式，没有人必须做所有的标签，并保持一些公平的概念。

# 7 任务

任务比练习更耗时，可以让你更深入地掌握内容。我们不期望你在实验室里完成所有的任务，但它们可以是很好的练习玩具问题。

1. [百科]针对查询q =[1, 2, 0, 0, 0, 0]，在以下一组测试文档上运行余弦相似度(可以是你自己的实现，也可以是上面两种中的一种):
   - $d_1$=[6、12、0、0、0]
   - $d_2$= [2,4,0,0,0,0]
   - $d_3$= [3,6,0,0,0,0]
   - $d_4$= [0,0,1,1,0,0]

   记下sim(q, d1)， sim(q, d2)， sim(q, d3)，和sim(q, d4)。你观察到什么，又如何解释?

2. [百科]重用之前实验室的电影评论数据，写一个程序，生成你选择的数据结构中的所有文档的词汇列表，包含以下信息:术语id，规范形式，逆文档频率。Term id应该只是一个术语的标识符，例如t001。规范形式可以被词干或词形化，任你选择。IDF是本文档中所示的标准公式。可选地，你的程序应该能够将该数据结构保存在文件中，并将其加载回来。

3.[百科]编写一个程序，创建一个词项-文档或文档-词项(无论你喜欢哪个，在那个阶段都不相关)矩阵，其中包含每个文档-词项对的词频值。程序应该可以选择使用二元词频权重、原始频率词频权重和对数比例频率词频权重。

4. [百科]编写一个程序，在控制台接收用户输入，并根据余弦相似度和你的词项权重方案搜索10个最相关的文档。你不仅要返回文档，还要返回一些文档ID。我建议使用文件名作为文档ID，以避免不得不决定命名方案。

5. [百科]如果可以，将自己与一个或多个随意的队友配对，并进行一些联合标签。商定一组3-5个你将在各自的搜索引擎上运行的查询，然后记录一个文档id以便分享你的结果。然后，在你们每个人之间分割这些文档id，并提供它们相关性的评估。最后，一旦你以(query, documententid, relevance)的形式记录了你的真实值，写一个程序来计算精度和召回率，并比较你的算法的性能。如果你不能或不想与另一个人配对，你可以通过简单地使用自己的搜索引擎的多种变体来做同样的事情，例如改变词的权重方案，或改变词汇表的大小，或改变词干提取/词形变换技术等。