

信息检索与数据挖掘实验三

- 19级人工智能班 张文浩 201900161140

实验题目

- IR Evaluation

实验环境

- window10、python3.6、spyder (anaconda)

实验内容

- 实现以下指标评价，并对Experiment2的检索结果进行评价
 1. Mean Average Precision (MAP): 平均精度值均值
 2. Mean Reciprocal Rank (MRR): 平均倒数排名
 3. Normalized Discounted Cumulative Gain (NDCG): 归一化折损累计增益

实验原理

• Mean Average Precision(MAP)：平均精度值均值

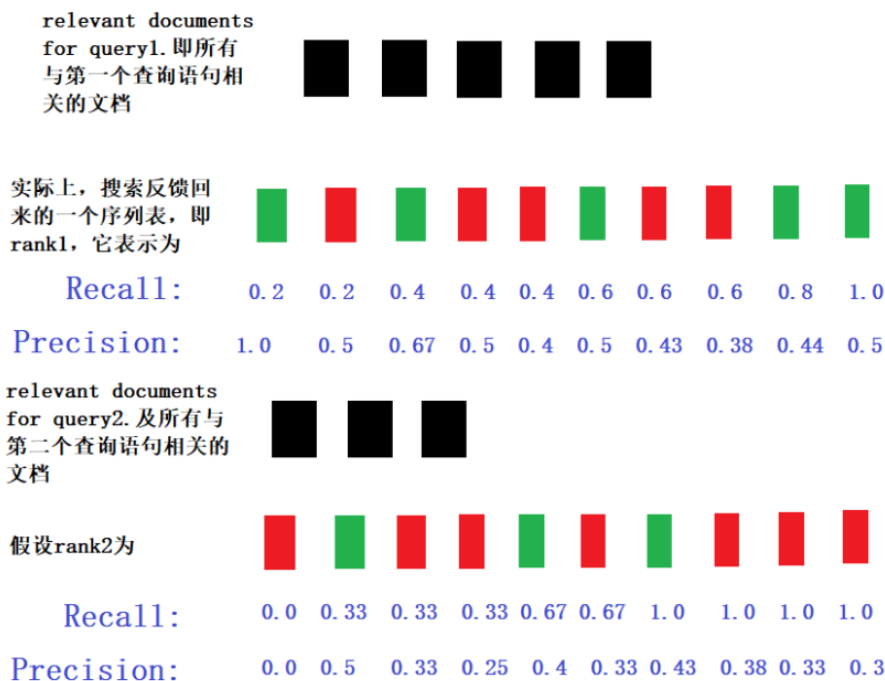
回顾Precision和Recall。

Precision：即精确率，检索最相关的顶级文档的能力。

Recall：即召回率，检索到语料库中所有相关项的能力。

精确率=检索到的相关文档数量/检索的文档总量， 召回率=检索到的相关文档数量/总的相关文档数量

• 例子



就拿rank1来讲，第一个返回的是正确结果，而且只有一个，所以

$\text{precision}=1, \text{recall}=1/5,$

因为有5个相关文档，此时已经检索到了1个，第二个红色，表示它其实不是相关文档，那么正确的还是只有一个，所以

$\text{recall}=1/5,$ 而 $\text{precision}=0.5,$

因为此刻返回来2个相关项，一个正确，一个错误，再看第三个，返回的是相关项，所以

$\text{recall}=2/5, \text{precision}=2/3.$

后面的按照此原理可以推导出来。接下来计算MAP

$\text{average precision query1}=(1.0+0.67+0.5+0.44+0.5)/5=0.62;$

$\text{average precision query2}=(0.5+0.4+0.43)/3=0.44;$

$\text{mean average precision}=(\text{ap1}+\text{ap2})/2=0.53;$

MAP即是算出所有查询相关项的精确度的平均值，再取所有查询的精确度的平均值。

• Mean Reciprocal Rank (MRR): 平均倒数排名

MRR得核心思想很简单：返回的结果集的优劣，跟第一个正确答案的位置有关，第一个正确答案越靠前，结果越好。具体来说：对于一个query，若第一个正确答案排在第n位，则MRR得分就是 $1/n$ 。（如果没有正确答案，则得分为0）

$$MRR = \frac{1}{|Q|} \sum_{i=1}^g \frac{1}{Rank_i}$$

• 例子

假设现在有5个query语句，q1、q2、q3、q4、q5

q1的正确结果在第4位，q2和q3没有正确结果（即未检索到文档），q4正确结果在第5位，q5正确结果在第10位。

那么系统的得分就是 $1/4+0+0+1/5+1/10=0.55$

最后MRR就是求平均，即该系统 $MRR=0.55/5=0.11$

• Normalized Discounted Cumulative Gain (NDCG): 归一化折损累计增益

DCG的两个思想：

- 1、高关联度的结果比一般关联度的结果更影响最终的指标得分；
- 2、有高关联度的结果出现在更靠前的位置的时候，指标会越高；

• 累计增益 (CG)

$$CG_n = \sum_{i=1}^n rel_i$$

- 折损累计增益 (DCG)

$$DCG_n = rel_1 + \sum_{i=2}^n \frac{rel_i}{\log_2 i}$$

- 关于DCG的计算，还有另一种比较常用的公式，用来增加相关度影响比重，计算方法为

$$DCG_n = \sum_{i=1}^n \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

- 归一化折损累计增益 (NDCG)

$$IDCG_n = rel_1 + \sum_{i=2}^{|REL|} \frac{rel_i}{\log_2 i}$$

或（因为老师上课讲的是上面的那一种，所以本次实验采用上面的计算方法）

$$IDCG_n = \sum_{i=1}^{|REL|} \frac{2^{rel_i} - 1}{\log_2(i + 1)}$$

则

$$NDCG_n = \frac{DCG_n}{IDCG_n}$$

- 例子

1. 计算CG


| n | doc # | relevance | |
|----|-------|-----------|-----------------|
| | | (gain) | CG _n |
| 1 | 588 | 1.0 | 1.0 |
| 2 | 589 | 0.6 | 1.6 |
| 3 | 576 | 0.0 | 1.6 |
| 4 | 590 | 0.8 | 2.4 |
| 5 | 986 | 0.0 | 2.4 |
| 6 | 592 | 1.0 | 3.4 |
| 7 | 984 | 0.0 | 3.4 |
| 8 | 988 | 0.0 | 3.4 |
| 9 | 578 | 0.0 | 3.4 |
| 10 | 985 | 0.0 | 3.4 |
| 11 | 103 | 0.0 | 3.4 |
| 12 | 591 | 0.0 | 3.4 |
| 13 | 772 | 0.2 | 3.6 |

2. 计算DCG

| n | doc # | rel | | log _n | DCG _n |
|----|-------|--------|-----------------|------------------|------------------|
| | | (gain) | CG _n | | |
| 1 | 588 | 1.0 | 1.0 | - | 1.00 |
| 2 | 589 | 0.6 | 1.6 | 1.00 | 1.60 |
| 3 | 576 | 0.0 | 1.6 | 1.58 | 1.60 |
| 4 | 590 | 0.8 | 2.4 | 2.00 | 2.00 |
| 5 | 986 | 0.0 | 2.4 | 2.32 | 2.00 |
| 6 | 592 | 1.0 | 3.4 | 2.58 | 2.39 |
| 7 | 984 | 0.0 | 3.4 | 2.81 | 2.39 |
| 8 | 988 | 0.0 | 3.4 | 3.00 | 2.39 |
| 9 | 578 | 0.0 | 3.4 | 3.17 | 2.39 |
| 10 | 985 | 0.0 | 3.4 | 3.32 | 2.39 |
| 11 | 103 | 0.0 | 3.4 | 3.46 | 2.39 |
| 12 | 591 | 0.0 | 3.4 | 3.58 | 2.39 |
| 13 | 772 | 0.2 | 3.6 | 3.70 | 2.44 |

3. 计算IDCG


| n | doc # | rel (gain) | CG _n | log _n | DCG _n |
|----|-------|------------|-----------------|------------------|------------------|
| 1 | 588 | 1.0 | 1.0 | 0.00 | 1.00 |
| 2 | 589 | 0.6 | 1.6 | 1.00 | 1.60 |
| 3 | 576 | 0.0 | 1.6 | 1.58 | 1.60 |
| 4 | 590 | 0.8 | 2.4 | 2.00 | 2.00 |
| 5 | 986 | 0.0 | 2.4 | 2.32 | 2.00 |
| 6 | 592 | 1.0 | 3.4 | 2.58 | 2.39 |
| 7 | 984 | 0.0 | 3.4 | 2.81 | 2.39 |
| 8 | 988 | 0.0 | 3.4 | 3.00 | 2.39 |
| 9 | 578 | 0.0 | 3.4 | 3.17 | 2.39 |
| 10 | 985 | 0.0 | 3.4 | 3.32 | 2.39 |
| 11 | 103 | 0.0 | 3.4 | 3.46 | 2.39 |
| 12 | 591 | 0.0 | 3.4 | 3.58 | 2.39 |
| 13 | 772 | 0.2 | 3.6 | 3.70 | 2.44 |



| n | doc # | rel (gain) | CG _n | log _n | IDCG _n |
|----|-------|------------|-----------------|------------------|-------------------|
| 1 | 588 | 1.0 | 1.0 | 0.00 | 1.00 |
| 2 | 592 | 1.0 | 2.0 | 1.00 | 2.00 |
| 3 | 590 | 0.8 | 2.8 | 1.58 | 2.50 |
| 4 | 589 | 0.6 | 3.4 | 2.00 | 2.80 |
| 5 | 772 | 0.2 | 3.6 | 2.32 | 2.89 |
| 6 | 576 | 0.0 | 3.6 | 2.58 | 2.89 |
| 7 | 986 | 0.0 | 3.6 | 2.81 | 2.89 |
| 8 | 984 | 0.0 | 3.6 | 3.00 | 2.89 |
| 9 | 988 | 0.0 | 3.6 | 3.17 | 2.89 |
| 10 | 578 | 0.0 | 3.6 | 3.32 | 2.89 |
| 11 | 985 | 0.0 | 3.6 | 3.46 | 2.89 |
| 12 | 103 | 0.0 | 3.6 | 3.58 | 2.89 |
| 13 | 591 | 0.0 | 3.6 | 3.70 | 2.89 |

4. 计算NDCG

| n | doc # | rel (gain) | CG _n | log _n | DCG _n |
|----|-------|------------|-----------------|------------------|------------------|
| 1 | 588 | 1.0 | 1.0 | 0.00 | 1.00 |
| 2 | 589 | 0.6 | 1.6 | 1.00 | 1.60 |
| 3 | 576 | 0.0 | 1.6 | 1.58 | 1.60 |
| 4 | 590 | 0.8 | 2.4 | 2.00 | 2.00 |
| 5 | 986 | 0.0 | 2.4 | 2.32 | 2.00 |
| 6 | 592 | 1.0 | 3.4 | 2.58 | 2.39 |
| 7 | 984 | 0.0 | 3.4 | 2.81 | 2.39 |
| 8 | 988 | 0.0 | 3.4 | 3.00 | 2.39 |
| 9 | 578 | 0.0 | 3.4 | 3.17 | 2.39 |
| 10 | 985 | 0.0 | 3.4 | 3.32 | 2.39 |
| 11 | 103 | 0.0 | 3.4 | 3.46 | 2.39 |
| 12 | 591 | 0.0 | 3.4 | 3.58 | 2.39 |
| 13 | 772 | 0.2 | 3.6 | 3.70 | 2.44 |



| n | doc # | rel (gain) | CG _n | log _n | IDCG _n |
|----|-------|------------|-----------------|------------------|-------------------|
| 1 | 588 | 1.0 | 1.0 | 0.00 | 1.00 |
| 2 | 592 | 1.0 | 2.0 | 1.00 | 2.00 |
| 3 | 590 | 0.8 | 2.8 | 1.58 | 2.50 |
| 4 | 589 | 0.6 | 3.4 | 2.00 | 2.80 |
| 5 | 772 | 0.2 | 3.6 | 2.32 | 2.89 |
| 6 | 576 | 0.0 | 3.6 | 2.58 | 2.89 |
| 7 | 986 | 0.0 | 3.6 | 2.81 | 2.89 |
| 8 | 984 | 0.0 | 3.6 | 3.00 | 2.89 |
| 9 | 988 | 0.0 | 3.6 | 3.17 | 2.89 |
| 10 | 578 | 0.0 | 3.6 | 3.32 | 2.89 |
| 11 | 985 | 0.0 | 3.6 | 3.46 | 2.89 |
| 12 | 103 | 0.0 | 3.6 | 3.58 | 2.89 |
| 13 | 591 | 0.0 | 3.6 | 3.70 | 2.89 |

实验步骤

准备工作

因为我在实验二中实现了所有smart检索方式，所以本次实验我用MAP、MRR、NDCG分别对

①Inc.ltn、②nnc.ntn、③anc.atn 进行评估主要对比计算不同term frequency计算方法对检索结果评分产生的影响。

| Term frequency | | Document frequency | | Normalization | |
|----------------|---|--------------------|---|--------------------|--|
| n (natural) | $tf_{t,d}$ | n (no) | 1 | n (none) | 1 |
| l (logarithm) | $1 + \log(tf_{t,d})$ | t (idf) | $\log \frac{N}{df_t}$ | c (cosine) | $\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$ |
| a (augmented) | $0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$ | p (prob idf) | $\max\{0, \log \frac{N - df_t}{df_t}\}$ | u (pivoted unique) | $1/u$ |
| b (boolean) | $\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$ | | | b (byte size) | $1/CharLength^\alpha$, $\alpha < 1$ |
| L (log ave) | $\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$ | | | | |

代码说明：

- 建立两个字典，qrels_dict存储助教给的qrels.txt里面的信息，用来查询每个query对每个doc的 relevance

```
qrels_dict = {} #字典用来存储输入的文件qrels.txt
qrels_real = {} #计算NDCG中要计算IDCG的ideal_score的字典
```

- 构造qrels_dict字典

```

qrels_dict = {}
with open(qrels_file, 'r') as f:
    for line in f:
        t = line.strip().split(' ')
        if t[0] not in qrels_dict:
            qrels_dict[t[0]] = {}
        if int(t[3]) > 0:
            qrels_dict[t[0]][t[2]] = int(t[3])

```

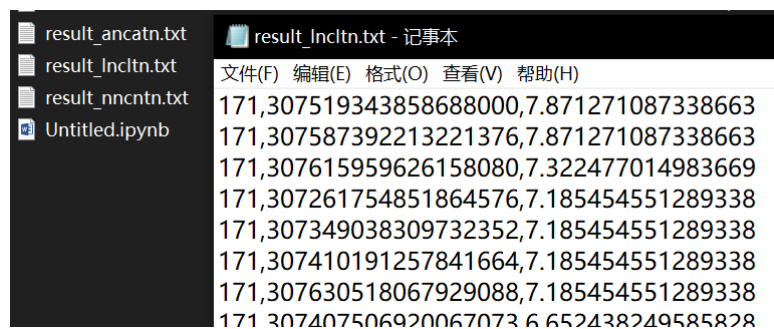
- qrels_dict长这样

```

{'171': {'307360182604820481': 2,
        '307575501373992961': 2,
        '307585630601371648': 2,
        '307592941277433856': 2,
        '307462808801513472': 2,
        '307581469876944897': 2,
        .....

```

- 我利用实验二分别利用三种方法进行了对queryid=171和172进行了查询，得到了三个结果文件。这三个文件长这样，第一列是queryid，第二列是docid，第三列是得分（在本次实验中用不到）



- 现在要读入这三个文件，分别存到三个字典中

```

result_dict_lnc1tn = {}
result_dict_nncntn = {}
result_dict_ancatn = {}
#以result_dict_lnc1tn为例
with open(result_file_lnc1tn, 'r') as f:
    for line in f:
        t = line.strip().split(',')
        if t[0] not in result_dict_lnc1tn:
            result_dict_lnc1tn[t[0]] = []
        result_dict_lnc1tn[t[0]].append(t[1])
.....
.....

```

- result_dict_lnc1tn长这样

```
{'171': ['307519343858688000',
        '307587392213221376',
        '307615959626158080',
        '307261754851864576',
        '307349038309732352',
        '307410191257841664',
        '307630518067929088',
        .....]
```

- 准备工作完成，现在使用MAP、MRR、NDCG计算得分

MAP

- 用三个数组分别存储三种检索方法对每一个query的AP (average precision) 得分。

```
AP_lnc1tn = [] #存储每一个query的average precision
AP_nnc1tn = []
AP_anc1tn = []
```

- 计算AP和MAP (以lnc1tn为例)

```
for q in result_dict_lnc1tn:
    result_q = result_dict_lnc1tn[q] #查询q得到的结果
    true_q = set(qrels_dict[q].keys()) #对于q真正相关的docid的集合
    precision = []
    i=0
    yes=0
    for docid in result_q:
        i+=1
        if(docid in true_q):
            yes+=1
            precision.append(yes/i)
    if yes:
        AP_lnc1tn.append(np.sum(precision)/yes)
MAP_lnc1tn = np.mean(AP_lnc1tn)
.....
.....
```

- 输出MAP评估的结果

```
lnc.ltn方法对两个query的查询的AP得分为:
[0.993010176011186, 0.8192010198546563]
nnc.ltn方法对两个query的查询的AP得分为:
[0.9751465981026507, 0.8084499136078966]
anc.ltn方法对两个query的查询的AP得分为:
[0.9928734507504109, 0.825429109566209]
lnc.ltn方法的MAP得分为:
0.9061055979329211
nnc.ltn方法的MAP得分为:
0.8917982558552737
anc.ltn方法的MAP得分为:
0.9091512801583099
```

MRR

- 用三个数组分别存储三种检索方法对每一个query的RR得分

```
RR_lnc1tn = []
RR_nncntn = []
RR_ancatn = []
```

- 计算RR和MRR (以lnc1tn为例)

```
for q in result_dict_lnc1tn:
    result_q = result_dict_lnc1tn[q] #查询q得到的结果
    true_q = set(qrels_dict[q].keys()) #对于q真正相关的docid的集合
    i = 0
    for docid in result_q:
        i+=1
        if docid in true_q:
            RR_lnc1tn.append(1/i)
            break
MRR_lnc1tn = np.mean(RR_lnc1tn)
.....
.....
```

- 输出MAP评估的结果

```
lnc.1tn方法对两个query的查询的RR得分为:
[1.0, 1.0]
nnc.ntn方法对两个query的查询的RR得分为:
[1.0, 1.0]
anc.atn方法对两个query的查询的RR得分为:
[1.0, 1.0]
lnc.1tn方法的MRR得分为:
1.0
nnc.ntn方法的MRR得分为:
1.0
anc.atn方法的MRR得分为:
1.0
```

NDCG

- 用三个数组分别存储三种检索方法对每一个query的NDCG得分

```
NDCG_each_lnc1tn = []
NDCG_each_nncntn = []
NDCG_each_ancatn = []
```

- 计算NDCG (以lnc1tn为例)

```
for q in result_dict_lnc1tn:
    result_q = result_dict_lnc1tn[q]
    true_q = list(qrels_dict[q].values())
    idea_list = sorted(true_q, reverse=True)
    i = 1
    DCG = qrels_dict[q].get(result_q[0], 0) #在每一个CG的结果上除以一个折损值
```

```

IDCG = idea_list[0]
length = len(idea_list)
for docid in result_q[1: length]:
    i += 1
    rel = qrels_dict[q].get(docid, 0) #.get后面两个参数，表示如果找不到docid，就
返回0
    DCG += rel / math.log(i, 2)
    IDCG += idea_list[i-1] / math.log(i, 2)
NDCG = DCG / IDCG #归一化处理
NDCG_each_lnc_ltn.append(NDCG)
NDCG_lnc_ltn = np.mean(NDCG_each_lnc_ltn)
.....
.....

```

- 输出MAP评估的结果

lnc.ltn方法对两个query的查询的NDCG得分为：
[0.8003863601998573, 0.7128927712359585]
nnc.ntn方法对两个query的查询的NDCG得分为：
[0.7791943855765212, 0.7073485912976111]
anc.atn方法对两个query的查询的NDCG得分为：
[0.8000657051883784, 0.7121355861855653]
lnc.ltn方法的NDCG得分为：
0.7566395657179079
nnc.ntn方法的NDCG得分为：
0.7432714884370661
anc.atn方法的NDCG得分为：
0.7561006456869719

对比

lnc.ltn方法的三种评价方法的得分分别是：
MAP: 0.906106 MRR: 1.000000 NDCG: 0.756640
nnc.ntn方法的三种评价方法的得分分别是：
MAP: 0.891798 MRR: 1.000000 NDCG: 0.743271
anc.atn方法的三种评价方法的得分分别是：
MAP: 0.909151 MRR: 1.000000 NDCG: 0.756101

实验总结

- 通过用三种评价方法对三种检索方法进行评估发现：
 - nnc.ntn的得分明显低于其他两种，说明计算term frequency的时候使用n (natural) 的方法效果是不如l (logarithm) 和a (augmented) 的。
 - 对于lnc.ltn和anc.atn两种检索方法而言，MAP评分anc.atn更高一点，NDCG评分lnc.ltn得分更高一点，但差别都不大，所以计算term frequency的时候使用l (logarithm) 和a (augmented) 的效果是差不多的。
 - 对于这三种评价方法而言，MRR的评估结果都是1，没有区分度，MAP和NDCG都还行，MAP的评估得分普遍更高一点（跟具体实验数据有关？）