

# 人工智能综合实践课程报告

侯新铭

2021201651

2022 年 9 月 4 日

## 1 整体介绍

*TF – IDF with classification* 是一个基于 *TF – IDF* 模型实现梯度分类优化的实时高效查询的搜索引擎；实现了从网页爬取、倒排索引建立到网页界面的一系列过程。项目基于[人大后勤集团](#)域名下全部约 9000 个网页，实现了实时搜索查询的功能。报告前半部分介绍了针对使用标准 *TF – IDF* 模型出现的问题作出的调整优化；后半部分介绍了 web UI 网页的构建与改善用户体验的功能实现。项目所有代码及相关文件开源在[Github](#)中。

## 2 搜索算法的优化

### 2.1 使用 idf 对 tf 加权

#### 2.1.1 这样加权有什么效果？

使得在计算 cosine score 时，奖励了包含于的文档个数更少的 term，更好地利用这些（文档层面的）低频词的 tf 来计算 score。

#### 2.1.2 负面效果和普适性评估？

一般无负面效果，有理论支持，普适性很强。

## 2.2 奖励包含不同关键词更多的网页

### 2.2.1 以此应对什么问题？

当我们分析采用标准 tf-idf 模型排序后的结果，发现排序很靠前的一些网页实际上包含的关键词“比例”并不高，其是因为个别词的 tf 很高才排到前面的；而对于普通用户，在其搜索构词中，关键词数相对少，其对自己输入的各个关键词都抱有的期待，更希望召回含有更多关键词的网页。

### 2.2.2 实现方式？

cosine score 计算过程里：外层循环遍历 query 里的关键词，内层循环对包含遍历到的关键词的网页加分 ( $tf * idf$ )；

不难看处，网页含有多少个关键词就“加了几次分”；因此，只要我们在每次加分时再加一个常数数，就实现了对包含更多关键词的奖励。

```
1 for q in query_terms:  
2     w_tq = tf_log_func(query_terms[q])  
3     postings_list = query_postings_list(q)  
4     for posting in postings_list :  
5         w_td = (tf_log_func(posting.tf)) * (posting . idf)  
6         scores [posting . docid] += w_td * w_tq + 3
```

此优化建立在标准 cosine score 计算模型上，对原本的 score 进行的调整实际上是离散的，含有相等不同关键词的网页获得的加分相同；我的模型中设置奖励常数为 1，相对 ( $tf * idf$ ) 来说很大，完全拉开了含不等数目的不同关键词的网页 score 间的差距，因此，本质上实现是一个梯度分类。这种分类对于“完全取词于同一网页，而没有基于用户需求进行构词”的评测来说肯定是有有效的，不会产生负面影响；

对于现实中用户自己进行构词的搜索来说，这样的分类弱化了 tf 的影响，可能会对检索产生负面效果；而参数的存在使得该优化具有更大的普适性，基于检索数据集交叉验证，对奖励参数大小进行调整，可能可以实现理想的效果。

## 2.3 奖励长度更长的网页

### 2.3.1 原因与实现方式？

尝试不对 cosine score 模型中的文档向量作归一化... 评测得分“居然”上升了许多，猜测是因为后勤集团域名下的网页的长短两极分化严重，而评测样例中最优网页大多为长网页。

### 2.3.2 普适性评估？

普适性无疑是差的，只是基于后勤集团域名下的网页特点和评测样例特点；且与 cosine score 相关理论相违背。

### 2.3.3 优化调整？

可以进一步尝试对网页长度取合适底数的对数，以适合数据集的权重考虑网页长度带来的影响。

## 2.4 奖励包含完整 query 的网页

### 2.4.1 以此应对什么问题？

通过上述优化，我们奖励了包含关键词更多的网页，但实际上我们没有规定这种包含是连续还是间断，倘若实现对“是否连续地包含所有关键词”的判断，我们的排序模型会更加稳健。

### 2.4.2 如何实现？

- 保存位置索引

先回顾一下我们实现 tf-idf 模型构造的倒排索引，可看作一个字典，key 是词，value 是“文档”构成的列表，这里的文档作为列表的元素，只包含 docid 序号是不够的，因此我们不妨构造一个新的数据类型（对应于课堂 ppt 中提到的 class Posting），其中打包保存了 docid, tf，以及 idf；而如若考虑“是否连续地包含关键词”，只保存这些参数是不够的，我们还需要一个由“文档内的位置索引”构成的列表。我们可以将切词后保存（词， docid）对的过程改进为保存（词， docid, 位置索引），而在后续构造倒排索引时将同一 docid 对应的位置索引构成一个列表，也存放到 Posting 中 docid, tf，以及 idf；

- 遍历与递归判断

在判断函数内，我们取第一个关键词，获得倒排索引字典中其对应的那个 value (posting 构成的列表)，先通过外层循环，遍历列表中 posting(docid)，再通过内层循环遍历对应的位置索引子列表中的位置，而后我们需要判断的是一种“匹配关系”，即“下一个词在该文档中是否出现在了上一个词出现位置的下一位”，这一关系的判断可以通过在“子判断函数”中递归调用“关键词在 query 中的位置”和“需要的位置索引”两个参数来实现；大致思路为，若递归期间出现“不匹配”的情况，返回 false，而若“关键词在 query 中的位置”递归调用到了 query 中关键词总数，返回 True。判断函数经过整个遍历过程，return “包含完整 query 的网页”构成的列表。

```
1 # 检查是否完整包含query函数 输出bool
2 def full_matching_phrase_docid(query):
3     ans_list = []
4     query_terms = [x for x in jieba.cut(query) if len(x) >= 1
5                   and x not in stopwords_set]
6     query_len = len(query_terms)
7     query_terms_docidlist = [query_docid(term) \
8                               for term in query_terms]
9     now_index = 1 #正在检验的词的index
10    #起始词的polistlist
11    term_polistlist = query_list_of_pos_list(query_terms[0])
12    for order, pos_list in enumerate(term_polistlist):
13        now_docid = query_terms_docidlist[0][order]
14        refer_pos_list = term_polistlist[order]
15        for pos in refer_pos_list:
16            if checknext(now_docid, now_index, query_terms, \
17                          query_terms_docidlist, refer_pos_list, pos):
18                if now_docid not in ans_list:
19                    ans_list.append(now_docid)
20            else:
21                continue
22    return ans_list
```

```
23 #在既定 docid 里，在既定 pos 处，后词的列表是否能对应上
24 def checknext(now_docid, now_index, query_terms,\ 
25 query_terms_docidlist, refer_pos_list, now_pos):
26     if now_index == len(query_terms):
27         return True
28     check_term_docid_list = query_terms_docidlist[now_index]
29     if now_docid not in check_term_docid_list:
30         return False
31
32     index_of_docid_in_purpose_term = query_terms_docidlist\
33 [now_index].index(now_docid)
34     check_pos_list = query_list_of_pos_list(query_terms\
35 [now_index])[index_of_docid_in_purpose_term]
36     purpose_pos = now_pos + 1
37     if purpose_pos in check_pos_list:
38         return checknext(now_docid, now_index+1, query_terms,\ 
39             query_terms_docidlist, refer_pos_list, purpose_pos)
40     else:
41         return False
```

- 基于此列表对 cosine score 模型计算出的分数加权

与前面所说的第二种优化类似，这一优化本质上也是分类，我们对列表内 docid 的 score 放大相同的倍数（因为这种分类极强，该倍数不妨比原来的 score 高出数个量级）。将倍增调整后的 score 重新排序，作为最终的 score。

#### 2.4.3 效果和普适性评估？

与优化 2 相同，因为放大的倍数相同，我们并没有改变同类网页之间的 cosine score 排序，因此不会对排序结果产生（除略微增加了时间复杂度外）的任何负面效果。我们通过对 cosine score 的调整来“标记分类”，相对于再另外设置 bool 变量来说，是一种更简单的实现方式。本质上，我们是将判断的“phrase（短语）是否出现在文档中”转化为了“短的词序列是否出现在长的词序列中”（这种转化基于切词方式的完全一致），受限于标点等细

节的差异与时间复杂度，直接判断前者是不可取的；这样的转化将“匹配”的范围略微扩大，使得匹配适应性有少量提升。这种优化“要求高”，十分适用于用户对 query 匹配要求高的情形。该优化直接了实现“必须包含某短语”需求的搜索；在此基础上，通过对用户的细化输入的获取，我们可以实现一些高级搜索（如必须包含某个或某几个关键短语）。

此优化中，我们检验的是是否返回完整的 query，为一个二分类；是否需要进一步调整函数，实现对网页中含有的 query 的子串的最长长度的标记呢？我觉得由于 query 本身的长度差异，连续性文本出现的价值也有差异；例如用户的 query 为一个单词加一个较短短语，对网页中比较短短语完整包含的奖励会冲击另一个同样重要的单词带来的影响。或许我们可以进一步尝试对 query 的长度进行判断，当且仅当 query 较长，且匹配子串占比较大的时候才进行奖励，这两个“较”到底要具体设置什么阈值有待基于数据集训练调整。

#### 2.4.4 对其他待实现优化的构想

- BM25

对网页长度作进一步考虑，作为 tf-idf 模型的升级，新增两个可调参数，可通过调参提升搜索引擎性能。

- 词距

弱化上述优化 4 中的“连续”，转而计算词距，即对 query 空间中关键词间词距和文本空间中出现的关键词间词距的关系作定性分析计算，基于得到的有效公式对分数进行加权。

## 3 Web UI 网页优化

### 3.1 美观性

- 背景图片，小图标，字体字号调整
- 鼠标悬浮时的效果
- 访问过的网页色彩变动
- 透明度、标题突出

- 对网页呈现对不同浏览器，不同设备的适应性
- 返回页面中当前搜索词呈现方式与位置

### 3.2 摘要选择与关键词标红显示

```
1 # 返回带有关键词标记的词列表
2 def query_abstract_for_url(query, url):
3     query_terms = [x for x in jieba.cut(query) if x != ' ']\ \
4         abstract_word_num = 90 #词数固定, len不固定
5     docid = get_key(url)
6     with open(os.path.join(mypath2,'{}{}.txt'.format(str(docid)))),\
7         encoding='UTF-8') as fin: #可以自动加||读
8         contents = fin.read()#只能read一次
9     words = [x for x in jieba.cut(contents) if x != '\n' and x !=\
10             '\r' and x != '\t' and x != ' ' and x != '\r\n' and x != '\n\r']\
11     contents_word_num = len(words)
12
13     max_contain = [(0,0)]
14     for start_word_num in range(max(0,contents_word_num)\ \
15         -abstract_word_num)):
16         window_words = []
17         for now_pick_num in range(abstract_word_num):
18             window_words.append(words[now_pick_num+start_word_num])
19
20         contain_num = 0
21         for term in query_terms:
22             if term in window_words:
23                 contain_num += 1
24             if contain_num == len(query_terms):
25                 max_contain = [(start_word_num, contain_num)]
26                 break
27             if contain_num >= max_contain[0][1]:
28                 max_contain = [(start_word_num, contain_num)]
```

```
29 real_start_num = max(0, max_contain[0][0] - 2)
30 real_end_num = min(contents_word_num, max_contain[0][0]\
+ abstract_word_num+2)
31 abstract_window = []
32 for k in range(real_start_num, real_end_num):
33     if words[k] in query_terms and words[k] not in stopwords_list:
34         abstract_window.append((words[k], 1))
35     else:
36         abstract_window.append((words[k], 0))
37 return abstract_window
```

```
1 <nobr width = 84px>
2 ...
3     {% for word in res['abstract'] %}
4     {% if word[1] == 1%}
5         <nobr style = "margin-right: -4px;color:red; \
6             font-size:15px;
7             text-align-last: center;">
8             {{word[0]}}
9         </nobr>
10    {% elif word[1] == 0%}
11        <nobr style = "margin-right: -4px;color:black; \
12            font-size:15px;text-align-last: center;">
13            {{word[0]}}
14        </nobr>
15    {% endif %}
16    {% endfor %}
17 ...
18 </nobr>
```

### 3.3 标题内容优化与去重

```
1 title = get_h1(url)
```

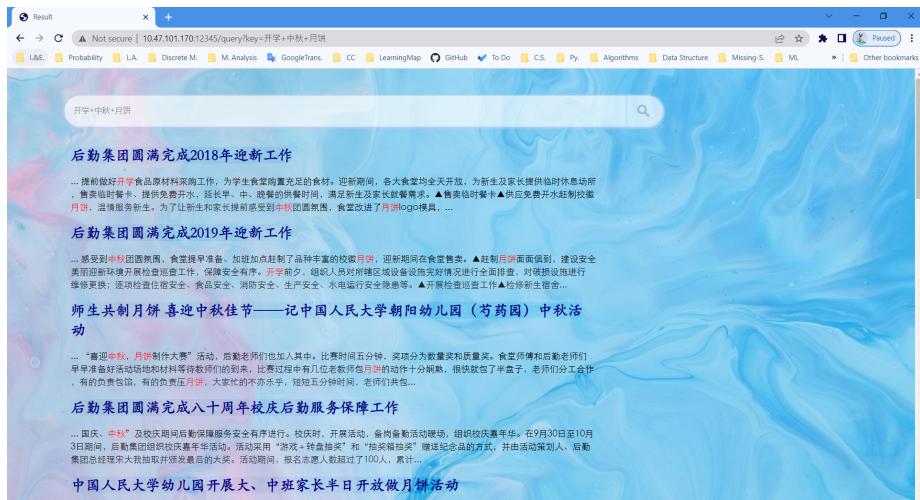
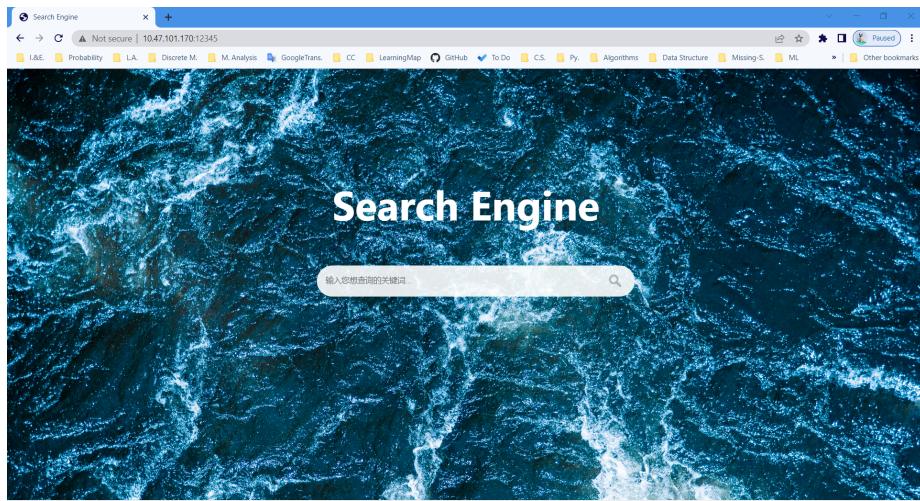
```
2     if title != None:
3         url_dict['title'] = title.text
4     else:
5         url_dict['title'] = get_title(url).text
6
7         url_dict['abstract'] = query_abstract_for_url(key, url) \
#tuple构成的列表
8         url_dict['url'] = url
9
10        if results != []: #用title排除了重复网页
11            if url_dict['title'] != results[-1]['title']:
12                results.append(url_dict)
13
14        results.append(url_dict)
```

### 3.4 对特殊搜索的适应性

- 空字符串
- 无效检索词
- 过长检索词
- 未找到任何含有关键词的网页

```
1 key = request.args.get('key') #用户发送的request里获取user输入key
2 key = key.replace(' ', '+')
3 terms =[x for x in jieba.cut(key) if len(x) >= 1 and x not \
4 in stopwords_set]
5 too_long = False
6 if terms == []:
7     return render_template('empty.html',too_long = too_long)
8 if len(terms)>30:
9     too_long = True
10    return render_template('empty.html',too_long=too_long)
```

### 3.5 页面展示



## 4 致谢

衷心感谢中国大学高阶人工智能学院毛佳昕老师的精彩授课与悉心指导，以及两位助教师兄的建议与帮助。