程序运行说明：

- **运行环境：**
  CPU：Intel(R) Xeon(R) CPU E5-2630 v4 @ 2.20GHz
  OS：Ubuntu 16.04.3 LTS
  Python 运行相关包见附录

- **运行步骤：**
  保证 data 文件夹下数据都在，输入命令 **source launch**，运行整个系统，包括进入运行环境（我们使用 virtualevn 建立本项目的虚拟环境，文件中已经包含虚拟环境 clinicalenv），启动 Elasticsearch，索引构建，模型训练，结果检索及排序，并获得最后的运行分数。

  注意：
  1. 需要使用 chmod 给 elasticsearch 下的文件增加 rw 权限，否则会出现 permission error
  2. 需要替换自己的 elasticsearch 可执行程序的路径，详细替换方法见 launch 脚本
  3. 在我们的服务器上启动 Elasticsearch 平均需要 30s，但由于服务器差异，在不同的机器上启动 Elasticsearch 时间不一样，若启动时间过长，可能会出现 Connection Error 提示，若出现这种提示，不比担心，只需等待 Elasticsearch 启动完毕后，重新 source launch 即可
  4. 最后执行 **source quit**，退出环境，关闭系统。
  5. data 文件夹下应该有病例文档数据 clinicaltrials_txt 和 clinicaltrials_xml！

- 结果文件：
  存放在 data 下，res*.txt 中

- Github: https://github.com/JamiesZhang/ClinicalTrials

- 有任何运行问题可邮件：shengsiyuan@ict.ac.cn or zhangjiahua@ict.ac.cn

现代信息检索 TREC PM Task2 大作业

小组成员：盛思远，张家华

摘要：Task2 旨在为病人找到符合条件的临床试验。我们使用 NCBI 的 E-utilities 的 ESearch-EFetch 方法进行查询扩展。借助 Elasticsearch 的 bulk API 并行批量建立索引。使用机器学习的方法进行排序学习，我们自己的新方法在 2.4 节中详细介绍。为了训练得到不同的检索模型、topic 字段和 doc 字段的阈值，我们使用了两种检索模型：VSM 和 BM25。topic 中使用了 disease 和 gene 两个字段。document 中使用了 brief_title 等六个字段。由于训练数据较少， 故需要选择 5 折交叉验证的方法。最终实验中，我们 P@5:0.4320, P@10:0.3786, P@15:0.3262。

1.引言

Precision Medicine（PM）主要是根据病人档案和生病历史为病人找到合适的治疗方法。由于现在有越来越多的关于 PM 的医学文献和治疗实验，使得医生根据简单的病历档案找到最合适的治疗方法和医学文献成为困难。所以使用信息检索的一些技术来快速的进行医学检索是 TREC 2017 的一个比赛主题。TREC 2017 比赛任务有两个： 1) 为病人找到最合适的医学治疗文献和 2)最符合病人情况的医学临床实验。本次大作业主要任务为解决 task 2。

```
- <topic number="1">
    <disease>Liposarcoma</disease>
    <gene>CDK4 Amplification</gene>
    <demographic>38-year-old male</demographic>
    <other>GERD</other>
  </topic>
```

图 1 topic 样例

在 TREC 2017 中，查询信息使用 topic 来表示，每一个 topic 包含 disease、gene、demographic 和 other 字段，分别表示病症，导致该病的基因（包括变异），年龄、性别等个人信息，其它病人可能包含的信息。TREC 2017 一共提供了 30 个 topic，每一个 topic 都代表一个病人信息，图 1 是其中一个 topic 样例。Clinical trail 任务的数据集是从 ClinicalTrials.gov 采集的快照。Task 2 的 Clinical Trial 任务是为病人找到符合条件的临床试验。

我们**系统架构**如图 2 所示，Elasticsearch 模块中包括使用 elasticsearch 建立索引和删除索引；Train 模块为训练模块；Preprocesss 模块是对数据的预处理模块，包括将 Topic 和病例 Document 转为 Elasticsearch 支持的 Json 数据格式，以及将训练数据划分成五折；TermExt 模块实现查询扩展功能；Search 模块包括查询及查询排序功能。

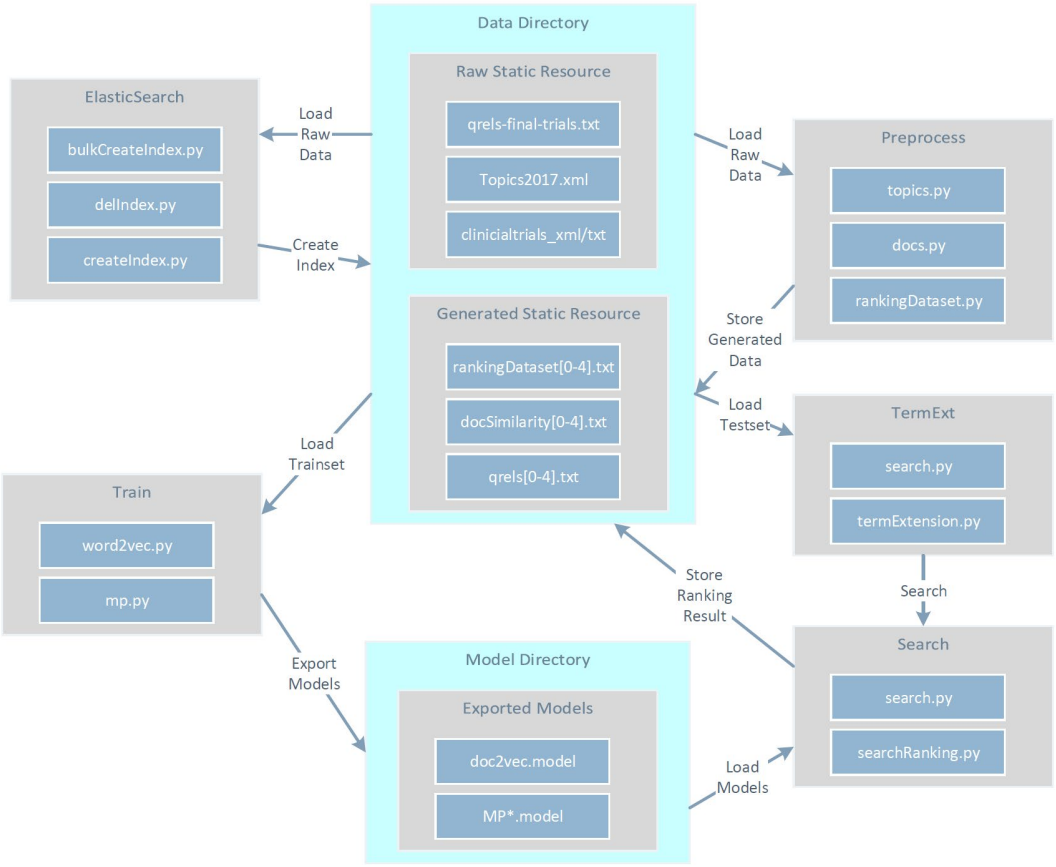

图 2 系统架构

2.具体实现

2.1 查询扩展（TermExt/termExtension.py）

我们首先对 Topics 中的词项进行扩展，主要针对 disease 域和 gene 域进行扩展。我们使用 NCBI 的 E-utilities 的 ESearch-EFetch[1]方法，当查询 disease 扩展词时，使用 MeSH 库；查询 gene 扩展词，使用 Entrez Gene library 库。

我们使用查询扩展的方法进行检索，得到的 P@10 结果效果不是很好，精确度低于不使用查询扩展的方法。经过分析，导致使用查询扩展精确度不增反降的可能原因有如下两方面：

1) 使用 NCBI 库在 Esearch 阶段，对于扩展词的查询会返回一系列相关词项 ID，根据 ID 又能查询一系列相关词项，如此多的相关词，由于专业限制，不明确医学数据，如何挑选更精确的同义词成为问题。（查询结果示例见图 3,4）

2) 我们的课本[3]在第九章 9.2.2 查询扩展一节也分析到，查询扩展通常可以提高召回率，而我们最后判断的结果是正确率，召回率高，精确率自然会有一定的下降。

所以最终提交结果为不使用查询扩展的原始 topic 进行查询。

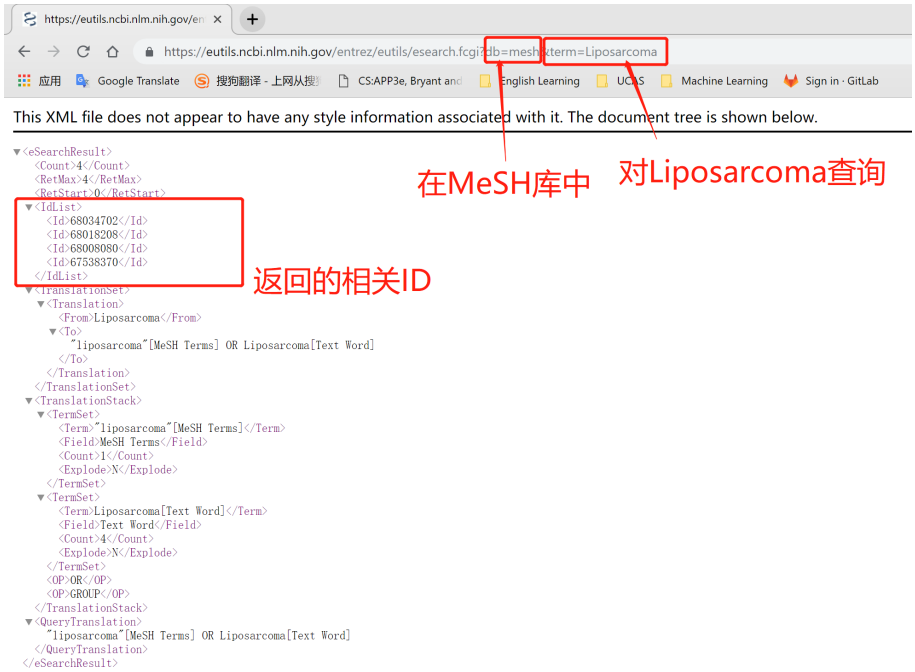

图 3 ESearch 获得相关词 ID

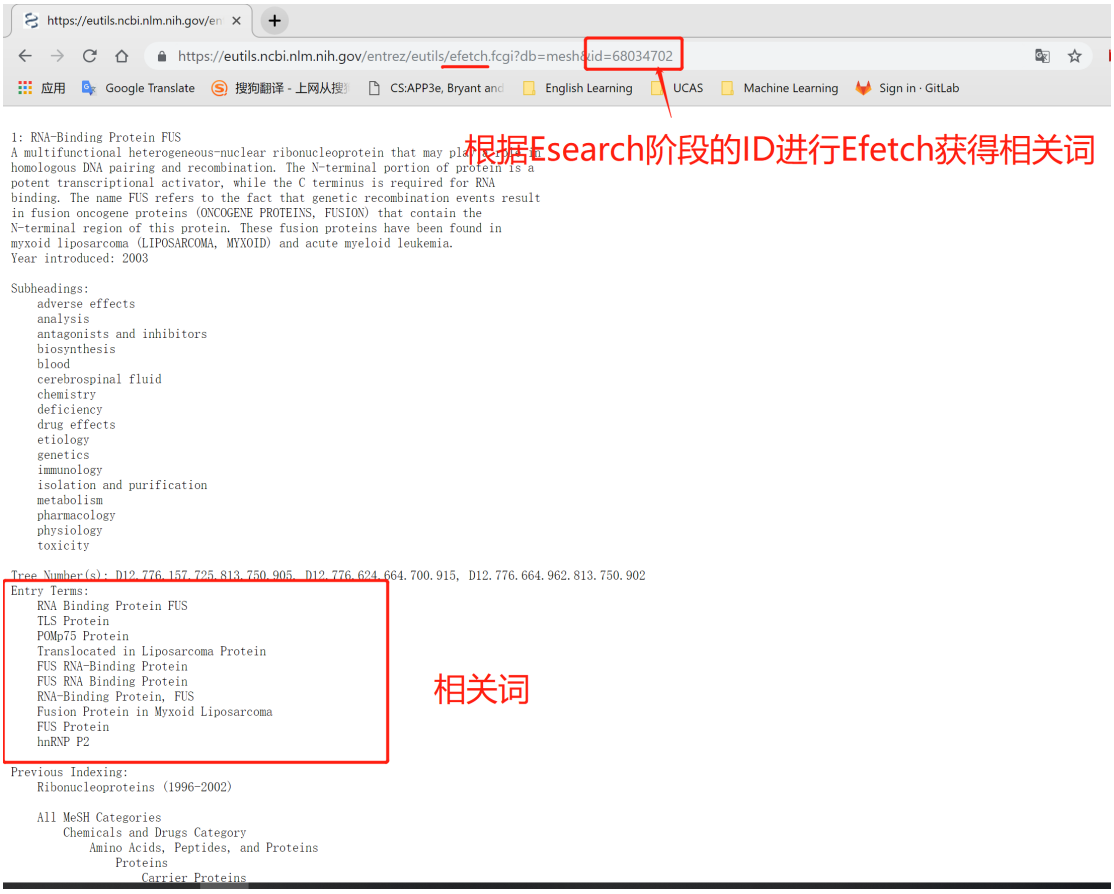

图 4 EFetch 过程根据相关 ID 获得扩展

2.2 搜索

索引建立和查询使用 Elasticsearch，Elasticsearch 是一个基于 Lucene 库的搜索引擎[2]，它直接提供了建立索引的方法和查询方法，便于使用。Elasticsearch 还提供了方便的 Python 接口，因此我们使用它提供的 Python 接口 elasticsearch-py 实现我们的程序。

Elasticsearch 提供了一整套基于 JSON 的 DSL 语言来定义查询。一般来说，会包含基本的 term 和前缀查询。同样也包含复合查询如 bool 查询。查询也可以设置过滤器，诸如 filtered。可以将 DSL 当作查询的一个抽象语法树（AST）。查询、过滤器以及设置之间可以互相包含，能够构建出各种复杂高级的查询。另外，Elasticsearch 还提供了许多内置的相似度模型，如向量空间模型 VSM（默认 tf-idf）以及 BM25。

通过 Elasticsearch 强大的功能，可以帮助我们快速的建立索引和实现查询，elasticsearch 也是蚂蚁金服内部最大的搜索平台。

2.2.1 Elasticsearch 的安装

具体安装参见官方安装步骤：https://www.elastic.co/downloads/elasticsearch。通过 pip install elasticsearch 命令安装响应的包。

安装完成后执行 bin/elasticsearch 启动 elasticsearch。

2.2.2 索引的建立（Search/bulkCreatIndex.py）

在为病例建立倒排索引时，我们挑选了如下几个重要的域进行检索分析，包括 nct_id、brief_title、official_title、brief_summary、study_type、primary_purpose、gender、minimum_age、maximum_age、healthy_volunteers、textblock（包括 brief_summary、detailed_description 和 eligibility）、mesh_term、condition、keyword。

在过滤器的设置上，我们使用了英文停用词 english_stop，和两种词干提取过滤器 light_english_stemmer 和 english_possessive_stemmer。

在此基础上，我们基于 BM25 和默认的 tf-idf 相似度计算模型建立了两个索引，在后面的工作中，我们会基于这两种相似度计算模型进行排序学习。

Elasticsearch 规定，索引一旦建立，不能对其进行修改，所以要想修改索引，必须先删除原有的索引，并重新建立，我们在(Search/delIndex.py) **中实现了索引的删除操作。**

起初我们建立索引由于没有使用并行化操作(Search/creatIndex.py)，导致建立所有文件的索引要用 6 个多小时，然后我们使用其提供的 bulk API，以 clinicaltrials_xml 下一级目录为一组数据，批量建立索引，效果有很大的提升，建立完整的索引大约平均只要 45 分钟，由于我们分别建立了 BM25 和 tf-idf 两个模型的索引，所以需要大约 1 个半小时的时间。

2.2.3 查询（Search/search.py）

我们的思路是通过机器学习的方法获得每个域的加权评分，所以在 Search/search.py 中通过 getScoreDic 函数获得针对某一个 topic 的 disease 域和 gene 域在 document 中所挑选出来的 6 个关键域的得分列表，以用于之后的权重训练。（如图 5 所示）

值得一提的是，Elasticsearch 本身并没有提供 bulk search 的 API。在预处理过程中我们需要对给定的 topic 和 doc 查询不同域上的评分，如果使用原始的 search 或者 index API，则会产生大量的通信开销。

对此我们通过 query API，对给定的 topic 在不同域上查询 top n 个相关的文档及其评分，如果所需的 doc 在返回结果中，就不需要额外的查询请求。这样就大大减少了预处理的时间。

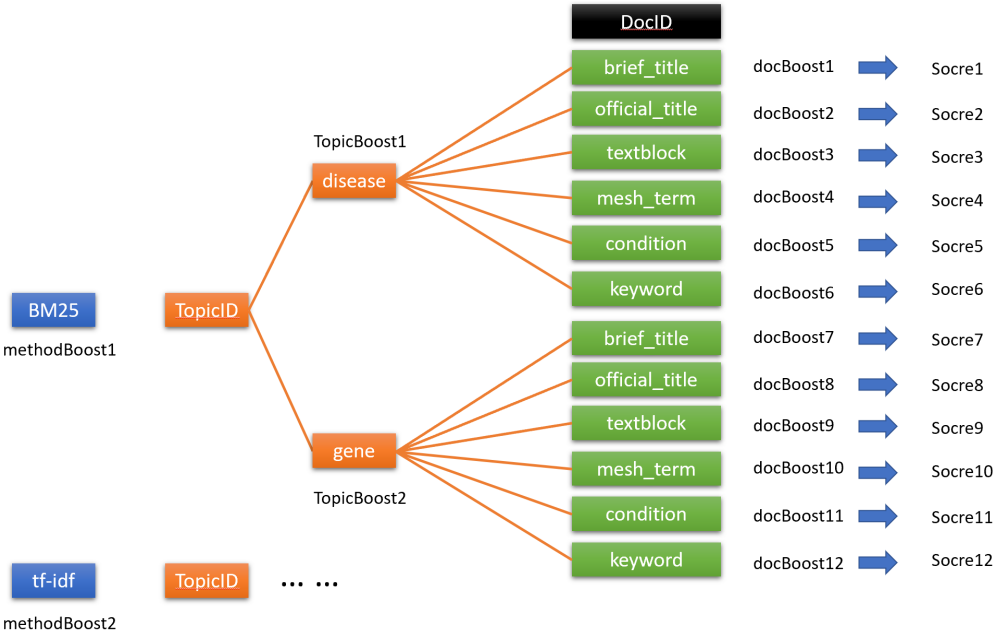

图 5 获得 Topic 域与 Document 域的单独得分

2.3 排序（Search/searhRanking.py）

在该模块中，输出最终的排序结果，并保存在 res*.txt 文件中。我们构造 baseBody 作为 elasticsearch 原始检索的查询 Body 字段，并根据经验设置其中域的一些权重。构造 queryBody 作为使用我们自己的模型检索的 Body 字段，其中的域权重使用我们使用机器学习训练（详细过程见下一章节）得到的权重。通过运行 getFinalResult 函数将最终的结果输出到 res*.txt 文件中。

2.4 模型训练

2.4.1 训练模块文件的说明

（1）train/word2vec.py：我们使用 gensim，对 clinical_txt 文件夹下的全部文档，建立了词项词典。通过 doc2vec，可以得到已有文档或者给定输入(given topic)的分布式表达。

该模型保存为 models/doc2vec.model，在预处理模块中被导入，用于得到给定 topic 和 doc 的相似度。并在后续的排序学习阶段被使用。

（2）train/mp.py：我们建立了如下的网络（图 6），通过排序学习的方式训练得到 Multilayer 的模型并导出。在最后的评测阶段，加载模型对测试集进行检索，并与 elasticsearch 的原始检索结果进行合并，从而得到最终的得分及排序（保存在 res*.txt）。

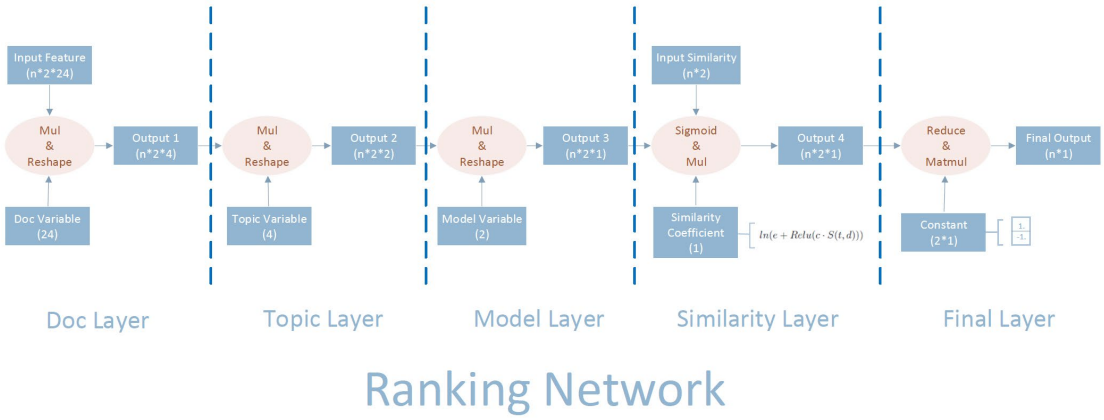

图 6 Ranking Network

2.4.2 基本思路

为了训练得到不同的检索模型、topic 字段和 doc 字段的阈值。我们使用了两种检索模型：VSM 和 BM25。topic 中使用了 disease 和 gene 两个字段。document 中使用了 brief_title 等六个字段。

我们在预处理模块中，已经对划分后的五折数据进行了逐个字段的评分查询，对每个(topcid, docid)的 pair，得到了 24 个评分，也就是 24 维的 feature vector。我们还对每个 pair 计算了分布式表达下的相似度。这些数据都会保存下来，以减少重复的数据处理过程。

由于使用了五折交叉验证，在排序学习阶段我们需要训练五个模型。对于每个模型，我们使用五折中的三折作为训练集。

导入事先保存好的训练集所对应的 feature vector 和相似度之后，将其输入到 TensorFlow 构建的网络当中。注意我们每个训练样本对应同一个 topic 下两个不

同 doc 的 feature vector，原因是我们使用的是排序学习，只需保证 largerDoc 评分大于 smallerDoc 评分的相对顺序即可。

对于输入的 feature vector，在第一层中与 doc field 的参数进行 element-wise 的乘法运算，然后每六个字段进行求和，即 reduce_sum，得到了 4 维的向量。这对应了两个检索模型下的两个 topic 字段的独立评分。

接下来的两层中，分别使用 topic field 和 model field 的参数进行类似的操作，最终得到了两个评分。即 largerDoc 和 smallerDoc 的评分。

由于参数可能是负权重导致得到负评分，我们使用了 sigmoid 的函数将评分转化成了 0 到 1 之间的正数。之后我们通过如图 7 所示的公式，将 word embedding 的结果与当前评分进行了合并，再做差即得到了最终的预测结果。显然，我们希望预测结果大于 0。

- Denote t as topicID, d as docID, $\Phi(t, d)$ returns the feature matrix $(2 \times 3 \times 6)$ of the given topic and document.
- W is the doc-field layer weight matrix with a shape of $(2 \times 3 \times 6)$
- V is the topic-field layer weight matrix with a shape of (2×3)
- W is the model-field layer weight matrix with a shape of (2)
- Denote c as the word2vec coefficient, and $S(t, d)$ returns the similarity based on word2vec between the given topic and document.

$$M = \Phi(t, d) \tag{1}$$

$$Temp(t, d) = \sum_i (\sum_j (\sum_k M_{ijk} \cdot W_{ijk}) \cdot V_{ij}) \cdot U_i \tag{2}$$

$$Score(t, d) = Sigmoid(Temp(t, d)) \cdot ln(e + Relu(c \cdot S(t, d))) \tag{3}$$

图 7 公式

最终训练得到的模型保存在 models/MP*文件夹下。

2.4.3 关于五折交叉验证

在 preprocess/rankingDataset.py 模块中，我们读入原始 qrels-final-trial.txt，根据 pdf 中的要求按照 topicid 划分成了五份，**保存为 qrels*.txt**。

对每份 qrels 中的每个(topicid, docid) pair，计算得到 doc2vec 的相似度以及 elasticsearch 不同检索模型、topic 字段和 doc 字段的 24 维评分向量。并保存为 rankingDataset*.txt 和 docSimilarity*.txt。

我们一共训练了五个模型，保存为了 MP*。对于模型 i，我们使用第 i 份作为验证集，第 i+1 份作为测试集，剩下三份作为训练集。

在排序学习过程中，我们分别加载相应的三份 rankingDataset.txt 和

docSimilarity.txt 作为训练输入。

在 evaluate 阶段，我们使用相应的一份 qrels 作为测试输入，并将最终的结果保存在 res*.txt 中。

3. 实验结果

Table 1: Eval Result

Run	Prec(5)	Prec(10)	Prec(15)
res0	0.4667	0.3333	0.3222
res1	0.4333	0.4000	0.2889
res2	0.4667	0.4333	0.3778
res3	0.3600	0.3600	0.3200
res4	0.4333	0.3667	0.3222
mean	0.4320	0.3786	0.3262

4. 总结

在实验的过程中，我们一开始将 topic 中的 other 字段也考虑了进去，得到的评分结果非常差，大约只有 0.1，在后期的调整过程中，无意间将 other 字段删除，瞬间得到了现在 0.37 的较好结果，着实有点坑。苦苦想了许久，竟然是因为多将 other 域考虑了进去，可见在查询中，并不是每一个域都是有用的，在这里 other 域就是一个巨大的噪声域，害人不浅。另外，已有的工具 Elasticsearch 也对我们系统实现提供了巨大的帮助，感谢 Elasticsearch。

还要感谢何苯老师以及《信息检索导论》，在这门课上我们学到了非常丰富的信息检索相关知识，相信在今后的学习和科研中能派上用场。

最后，祝愿老师及助教新年快乐，在 2019 年科研顺利，硕果累累，身体健康，万事如意！

参考文献：

[1]https://www.ncbi.nlm.nih.gov/books/NBK25498/#chapter3.ESearch__ESummaryEFetch

[2]Elasticsearch. https://zh.wikipedia.org/wiki/Elasticsearch

[3] Christopher D.Manning. 现代信息检索[M].人民邮电出版社，2018. 130.

附录：

我们的系统中使用到了如下 python 包，若您的电脑上若没有可能需要安装：

absl-py 0.6.1

astor 0.7.1

boto 2.49.0

boto3	1.9.71
botocore	1.12.71
bz2file	0.98
certifi	2018.11.29
chardet	3.0.4
docutils	0.14
elasticsearch	6.3.1
gast	0.2.0
gensim	3.6.0
grpcio	1.17.1
h5py	2.9.0
idna	2.8
jmespath	0.9.3
Keras-Applications	1.0.6
Keras-Preprocessing	1.0.5
lxml	4.2.5
Markdown	3.0.1
numpy	1.15.4
pip	18.1
protobuf	3.6.1
python-dateutil	2.7.5
requests	2.21.0
s3transfer	0.1.13
scipy	1.2.0
setuptools	40.6.3
six	1.12.0
smart-open	1.7.1
tensorboard	1.12.1
tensorflow	1.12.0
termcolor	1.1.0
urllib3	1.24.1
Werkzeug	0.14.1
wheel	0.32.3