南開大學
Nankai University

南　开　大　学

计 算 机 学 院

信息检索系统实验报告

---

hw3-enron_mail_searcher

姓名：王泳鑫

学号：1911479

年级：2019级

专业：计算机科学与技术

指导教师：温延龙

2021 年 11 月 21 日

摘要

关键字：倒排索引，**index**，**elasticsearch,enron**

目录

一、 实验描述 **1**

二、 实验设计与代码实现 **1**

三、 实验演示与结果分析 **4**

四、 探索**ES** **7**

一、 实验描述

基于ElasticSearch + Python开发环境，对安然公司150位用户的50W封电子邮件进行检索系统实现。

hint：

1. 可以按照收件人、发件人、标题、内容等进行邮件检索
2. 探索ES实现索引构建、向量空间模型等核心环节
3. 可以提取附件内容，进行附件检索
4. 垃圾邮件分类
5. 文本情感分析
6. GUI、Web呈现检索系统
7. 基于已学设计更多内容

二、 实验设计与代码实现

本次实验时基于ElasticSearch和Python开发环境，实现对enron邮件数据集的索引建立与检索，同时，应用了ES的可视化插件elasticsearch-head来协助检索工作。

首先需要建立索引，借助es.indices.create函数，我实现了一个创建索引的函数create_index，通过es.indices.exists判断，如果已经用索引占用了这个名字，则使用es.indices.delete删掉这个索引，同时新创建一个新索引，然后调用我们创建的put_map函数。

```
INDEX_NAME = 'enron-email'

es = Elasticsearch()

p = Parser()

def create_index():
    if es.indices.exists(INDEX_NAME):
        es.indices.delete(INDEX_NAME)
        print('删除存在的索引 \'{}\' ，并创建一个新的索引'.format(INDEX_NAME))
    result = es.indices.create(index=INDEX_NAME, ignore=400)
    #print(result)
    put_map()
    return
```

对于put_map函数，我们需要创建索引的结构，在body里面，我把dynamic设置为strict，如果遇到新的字段就抛出异常，同时，我们的_source字段默认时存储的，但是对于某个字段的内容非常多，只需要返回文档id的情况，我们就可以无需保留_source字段，这样能节省很多很多空间，如果只想存储某几个字段，我们可以通过includes参数来设置，例如：

```
    body=
        {
            "dynamic": "strict",
            "_source": {"includes":["from","to"]},
            "properties":
```

```
                {
                    "content-transfer-encoding": {"type": "text"}
                    ,"message_body": {"type": "text"}
                    ,"content-type": {"type": "text"}
                    ,"x-bcc": {"type": "text"}
                    ,"from": {"type": "keyword"}
                    ,"x-from": {"type": "text"}
                    ,"x-filename": {"type": "text"}
                    ,"x-folder": {"type": "text"}
                    ,"to": {"type": "keyword"}
                    ,"x-to": {"type": "text"}
                    ,"mime-version": {"type": "keyword"}
                    ,"cc": {"type": "text"}
                    ,"x-cc": {"type": "text"}
                    ,"bcc": {"type": "text"}
                    ,"x-bcc": {"type": "text"}
                    ,"subject": {"type": "text"}
                    ,"message-id": {"type": "keyword"}
                    ,"x-origin": {"type": "text"}
                    #,"date": {"type": "date", "format": "EEE, dd MMM yyyy HH:mm:ss Z
                                                                    (z)"}
                    , "date": {"type": "keyword"}
                    ,"att":{"type":"keyword"}
                }
        }
```

这样我们就可以只存储“from”和“to”字段，这些字段中，date字段要尤为注意，如果按照数据集里面时间的格式，有一些可以正常存储，而一少部分会报错，所以我暂时用keyword来替代，这样就无法实现一个时间段区间邮件的检索了。通过一些资料查询，

```
def put_map():
    es.indices.put_mapping(
        body=
        {
            "dynamic": "strict",
            #"_source": {"enabled": "false"},
            "properties":
                {
                    "content-transfer-encoding": {"type": "text"}
                    ,"message_body": {"type": "text"}
                    ,"content-type": {"type": "text"}
                    ,"x-bcc": {"type": "text"}
                    ,"from": {"type": "keyword"}
                    ,"x-from": {"type": "text"}
                    ,"x-filename": {"type": "text"}
                    ,"x-folder": {"type": "text"}
                    ,"to": {"type": "keyword"}
                    ,"x-to": {"type": "text"}
                    ,"mime-version": {"type": "keyword"}
                    ,"cc": {"type": "text"}
                    ,"x-cc": {"type": "text"}
                    ,"bcc": {"type": "text"}
                    ,"x-bcc": {"type": "text"}
                    ,"subject": {"type": "text"}
                    ,"message-id": {"type": "keyword"}
                    ,"x-origin": {"type": "text"}
```

```
                    #,"date": {"type": "date", "format": "EEE, dd MMM yyyy HH:mm:ss Z
                                                                (z)"}
                    , "date": {"type": "keyword"}
                    ,"att":{"type":"keyword"}
                }
        }
        ,index=INDEX_NAME
        ,doc_type='enron-type'
        ,include_type_name=True
    )
```

然后，我们就需要遍历文件夹，对每个文件建立索引，代码如下：

```
def data_read():
    mail_dir = 'D:/Data1/maildir'
    create_index()
    prefix_size = len(mail_dir) + 1

    for root, dirs, files in os.walk(mail_dir, topdown="false"):

        for filename in files:
            nameOfFileToOpen = "{0}/{1}".format(root, filename)
            contents = get_enron_eml_content(nameOfFileToOpen)
            index_into_elasticsearch(nameOfFileToOpen,  filename, contents)
```

通过mail_dir输入enron数据集的路径，调用创建索引的函数，然后通过os.walk函数遍历整个文件夹，os.walk返回的是一个3个元素的元组 (root, dirs, files)，分别表示遍历的路径名，该路径下的目录列表和该路径下文件列表。其中topdown设置为false，表示优先遍历根目录下的子目录，是一种自底向上遍历的方式。

然后在对files中的每一个文件进行文件的读取，通过get_content函数和index_into_elasticsearch函数载入ES，代码如下：

```
def get_enron_eml_content(eml_file_to_open):
    data_file = open(eml_file_to_open,encoding='gbk',errors='ignore')
    contents = ""
    try:
        for line in data_file:
            contents += line
    finally:
        data_file.close()
    return contents


def get_att(eml_file_to_open):
    s=" - "
    s=s+r'(.+?)\.'
    data_file = open(eml_file_to_open,encoding='gbk',errors='ignore')
    tt=data_file.readlines()
    lastline = tt[-1]
    m=re.findall(s,lastline)
    if len(m) != 0:
        return str(m)
    else:
        return " "
```

其中，get_enron_eml_content函数用来得到一篇邮件的内容，其中文件的open函数要加上encoding='gbk'和errors='i
解决由于utf-8编码出错的问题，同样地，get_att函数用来获取附件，因为附件总在一篇邮件的最后一行，因此我们readlines后直接通过索引-1来得到邮件的最后一行，然后通过正则表达式匹配来得到附件名称。

然后再把索引载入ES中，代码如下：

```
def index_into_elasticsearch(nameOfFileToOpen, filename, contents):
    msg = p.parsestr(contents)
    jsonMapDoc = {}

    headers = dict(msg._headers)
    for key, value in headers.items():
        key = key.lower()
        if not value.find(",") == -1 and key != "date" and key != "subject":
            value = value.split(",")
            jsonMapDoc[key] = value
        else:
            jsonMapDoc[key] = value

    jsonMapDoc["message_body"] = msg._payload
    jsonMapDoc["att"] = get_att(nameOfFileToOpen)
    file_size = os.path.getsize(nameOfFileToOpen)
    try:
        es.index(index=INDEX_NAME, doc_type="enron-type", body=jsonMapDoc)
    except Exception as ex:
        traceback.print_exc()
        print("Failed to index the document {}".format(jsonMapDoc))
    return
```

Parser中的parsestr函数可以把邮件的内容变成词典，然后就可以通过key和value来给jsonMapDoc赋值，其中对于“to”、“x-to”等字段可能由多个值所以我们就需要使用split函数将它们且分开，再存入jsonMapDoc中，对于“att”字段，直接调用get_att函数，然后把jsonMapDoc存入索引之中。

三、 实验演示与结果分析

当索引已经完全载入ES时，我们可以通过插件elasticsearch-head来查看索引的状态，如图2所示

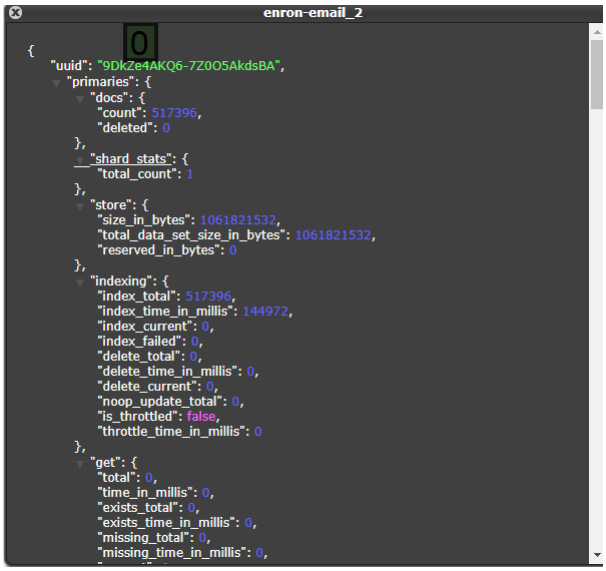

图 1: 索引状态

然后我们可以基本的es.search函数来检索邮件内容，例如：

```
    query = {
     "query":{
          "bool":{
               "must":[{
                    "prefix":{
                         "from":"susan"
                    }
               }
               ],
               "must_not":[],
               "should":[]
          }
     },
     "from":0,
     "size":10,
     "sort":[],
     "aggs":{}
}

res = es.search(index='enron-email_2',body=query)
print(res)

output:
{'took': 3, 'timed_out': False, '_shards': {'total': 1, 'successful': 1, 'skipped': 0,
                                            'failed': 0}, 'hits': {'total': {'value':
                                            5473, 'relation': 'eq'}, 'max_score': 1.0,
                                            'hits': [{'_index': 'enron-email_2', '_type
                                            ': 'enron-type', '_id': '
                                            2i1ZQH0BdQ4krPrXDx4q', '_score': 1.0, '
                                            _source': {'message-id': '<6945855.
                                            1075855697249.JavaMail.evans@thyme>', 'date
                                            ': 'Tue, 20 Feb 2001 23:04:00 -0800 (PST)',
                                             'from': 'phillip.allen@enron.com', 'to': '
                                            jacquestc@aol.com', 'subject': 'Re: General
```

```
 Issues', 'mime-version': '1.0', 'content-type': 'text/plain; charset=us-ascii', 'content-transfer-encoding': '7bit', 'x-from': 'Phillip K Allen', 'x-to': 'JacquesTC@aol.com @ ENRON', 'x-cc': '', 'x-bcc': '', 'x-folder': '\\Phillip_Allen_June2001\\Notes Folders\\All documents', 'x-origin': 'Allen-P', 'x-filename': 'pallen.nsf', 'message_body': 'That would we very helpful.  \n\nThanks,\n\nPhillip', 'att': ' '}}]}}
```

可以来测试我们的附件检索功能，例如我们检索附件名为DRAW2的文件：

```
    qq={
     "query": {
          "bool": {
               "must": [
                    {
                    "match": {
                    "att": "['DRAW2']"
                    }
               }
          ],
          "must_not": [ ],
          "should": [ ]
          }
     },
     "from": 0,
     "size": 10,
     "sort": [ ],
     "aggs": { }
}

res = es.search(index='enron-email_2',body=qq)

print(res)

output:

Subject: DRAW2.xls\n\n\nEnclosed is a copy of one of the draws submitted to Bank One for a prior\njob.\n\nGeorge W. Richards\nCreekside Builders, LLC\n\n\n - DRAW2.xls\n', 'att': "['DRAW2']"
```

我们可以通过插件elasticsearch-head来检验检索的正确性，如图2所示

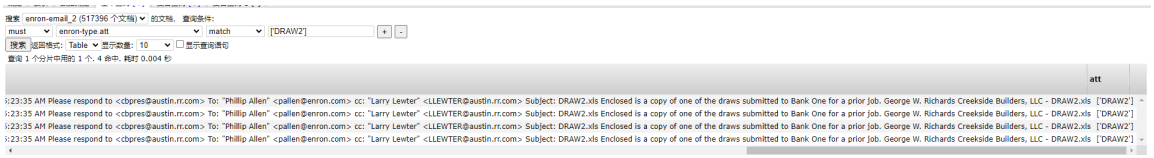

图 2: 检索结果

四、 探索ES

在程序进行的时候，也就是创建索引时，我打开插件elasticsearch-head看到索引状态，有时候doc的数值会停止增长，但是index仍然不断增加，我很好奇这其中的原理，经过学习，我发现ES在写入数据时，会有以下八个阶段：

1. 先写入buffer，在buffer里的时候数据是搜索不到的；同时将数据写入translog日志文件;
2. 如果buffer快满了，或者到一定时间，就会将buffer数据refresh到一个新的segment file中，但是此时数据不是直接进入segment file的磁盘文件的，而是先进入os cache的。
3. 只要数据进入os cache，此时就可以让这个segment file的数据对外提供搜索了;
4. 重复1 3步骤，新的数据不断进入buffer和translog，不断将buffer数据写入一个又一个新的segment file中去，每次refresh完buffer清空，translog保留。随着这个过程推进，translog会变得越来越大。当translog达到一定长度的时候，就会触发commit操作。
5. commit操作发生第一步，就是将buffer中现有数据refresh到os cache中去，清空buffer;
6. 将一个commit point写入磁盘文件，里面标识着这个commit point对应的所有segment file;
7. 强行将os cache中目前所有的数据都fsync到磁盘文件中去;
8. 将现有的translog清空，然后再次重启启用一个translog，此时commit操作完成.