

QINMIN HU

2018年6月30日

Final Project

一. 实验目的

1. 给定topic及某个引擎检索的结果，进行重新排序。
2. Learning-to-rank，同样的数据集：
 - 2.1.简单的Point-wise实现
 - 2.2.复现已有的LTR算法并比较/改进现有算法
3. 可以调用相关工具包

二. 实验数据

1. trainFolder=[]#训练集的doc文件夹名,即对应的query文件的名字
2. testFolder=[]#测试集的doc文件夹名,也是对应的query的名字
3. trainQrels={}#记录每个querylabel1的docname
4. stopwords=[]#停用词
5. docLength={}#文档长度
6. wordDir={}#倒排索引，word是key
7. wordDocFreq={}#包含单词的文档数

三. 实验步骤

1. 生成训练集

1.1.加载 query 和 document

(加载查询文件中的 **title** 作为 **query**，加载相应的 **document** 的 **title** 和 **abstracttext** 作为这个 **text** 的内容)

```
def loadXmlText(filename,docdir,foldername):  
    if filename.endswith('.xml'):  
        xmldoc=ET.parse(docdir+'/'+foldername+'/'+filename)
```

```

else:
    os.rename(docdir+'/' + foldername + '/' + filename, docdir + '/' +
foldername + '/' + filename + '.xml')
    xmldoc = ET.parse(docdir + '/' + foldername + '/' + filename +
'.xml')

test=""
for MedlineCitation in xmldoc.findall('MedlineCitation'):
    for article in MedlineCitation.findall('Article'):

        for journal in article.findall('Journal'):

            for title in journal.findall('Title'):

                test = test + ' ' + title.text
            for a in article.findall('Abstract'):
                for abstract in a.findall('AbstractText'):

                    if abstract.text==None:
                        for s in abstract.findall('AbstractText'):
                            test = test + ' ' + s.text
                    else:
                        test = test + ' ' + abstract.text

testList=[]

i=0
for word in test.split():

    if re.findall(r'\d+', word) or word == "":
        continue
    while '_' in word or '"' in word or '.' in word or '-' in word or '!' in
word or ';' in word or ',' in word or '$' in word or '/' in word or '/' in word or '{' in
word or '}' in word or '*' in word or '#' in word or '^' in word or '|' in word or '~' in
word or '=' in word or '\' in word or '+' in word or ':' in word or '?' in word:
        word = word.replace("_", "").replace('"', "").replace(".",
""),).replace('-', "").replace("!", "").replace(
";",
""),).replace(
",", "").replace("$", "").replace("//", "").replace(/,
""),).replace('{', "").replace('}', "").replace(
'*',
""),).replace(
'^', "").replace("|", "").replace('~', "").replace('=',
""),).replace("\", "").replace('+', "").replace(

```

```

        ':',
        "").replace(
        '?', "").replace("#", "")
    testList.append(word)

```

1.2.预处理

(词条化)

```

def tokenization(str_test):
    word_token=[]
    for i in range(len(str_test)):
        if str_test[i]==None:
            word_token.append("None")
        elif re.findall(r'\S', str_test[i]):

            word_token.append(nltk.word_tokenize(str_test[i])[0])
    return word_token

```

(去除停用词)

```

def stopword(str_test):
    test=[]
    for word in str_test:
        if word in stopwords:
            continue
        else:
            test.append(word)
    return test

```

(词形归并)

```

def lemmatization(str_test):
    test=[]
    lemmatizaer=WordNetLemmatizer()
    for i in range(len(str_test)):
        if str_test[i]=="None":
            continue
        else:
            test.append(lemmatizaer.lemmatize(str_test[i]))
    return test

```

(词干还原)

```

def stemmed(str_test):
    test=[]
    porter_stemmer = PorterStemmer()
    for word in str_test:
        test.append(porter_stemmer.stem(word))
    return test

```

```

text_token=tokenization(testList)
text_stopword=stopword(text_token)
text_lemmatize=lemmatization(text_stopword)
text=stemmed(text_lemmatize)

```

1.3.倒排索引

```

def loadDocument(foldername,docdir,query):
    docLength={}#文档长度
    wordDir={}#倒排索引，word是key
    N=0
    for root,dirs,files in os.walk(docdir+'/'+foldername):
        count=0
        for file in files:
            if file.endswith('.gz'):
                continue
            else:
                count=count+1
        for file in files:
            if file=='.DS_Store':
                continue
            if file.endswith('.gz'):
                continue
            else:
                N=N+1
            if re.findall(r'\d+', file):
                text=loadXmlText(file,docdir,foldername)
                docLength[file]=len(text)
                wordFreq={}
                for word in text:
                    if word in wordDir:
                        if file in wordDir[word]:
                            wordDir[word]
                                [file]=wordDir[word][file]
                        else:
                            wordDir[word][file]=1
                    else:
                        test={}
                        test[file]=1
                        wordDir[word]=test

```

1.4.计算 tfidf、bm25

```

def getTfidf(query,wordDir,N,docLength):

```

```

wordDocFreq={}
score={}
for key,value in wordDir.items():
    wordDocFreq[key]=len(value)
for doc in docLength.keys():
    s=0
    for word in query:
        if word in wordDocFreq.keys():
            idf=math.log(N/wordDocFreq[word])
        else:
            idf=0
        if word in wordDir.keys():
            if doc in wordDir[word].keys():
                freq=wordDir[word][doc]/docLength[doc]
                s=s+idf*freq
            else:
                s=s+0
    score[doc]=s
return score

def getBM25(query,wordDir,N,docLength):
    wordDocFreq={}
    score={}
    for key,value in wordDir.items():
        wordDocFreq[key]=len(value)
    count=0
    for doc in docLength.keys():
        count=count+docLength[doc]
    avgLen=count/N

    for doc in docLength.keys():
        s=0
        for word in query:
            if word in wordDocFreq.keys():
                idf=math.log((N-wordDocFreq[word]+0.5)/(0+0.5))
            else:
                idf=math.log((N-0+0.5)/(0+0.5))
            if word in wordDir.keys():
                if doc in wordDir[word].keys():
                    freq=wordDir[word][doc]
                    s=s+idf*freq*2.5/
(freq+1.5*(0.25+0.75*docLength[doc]/avgLen))
                score[doc]=s
    return score

```

1.5.写入训练集文件

(将 query 文件名、documentname、tfidf、tfidf 的排名、bm25、bm25 的排名、对应的 label 写入 10152130138Traindata 文件)

```
def outputFile(tfidfSort,bm25Sort,queryname):

    f=open('Traindata','a')

    bm25KeyList=[]
    bmDic={}

    for key,value in bm25Sort:
        bmDic[key]=value
        bm25KeyList.append(key)
    tfidfIndex=0

    for key,value in tfidfSort:
        tfidfIndex=tfidfIndex+1
        keylist=key.replace('.xml','')
        keyIndex = bm25KeyList.index(key) + 1
        bmS = bmDic[key]
        if keylist in trainQrels[queryname]:

            writeStr=str(queryname) + ' '+str(keylist)+' ' +str(tfidfIndex)
            +' '+str(keyIndex)+' 1\n'
        else:
            writeStr = str(queryname) + ' ' + str(keylist) + ' ' +
            str(tfidfIndex) + ' ' + str(keyIndex) + ' 0\n'
        f.write(writeStr)
    f.close()
```

2. 生成测试集（同训练集）

2.1.加载 query 和 document

2.2.预处理

2.3.倒排索引

2.4.计算 tfidf、bm25

2.5.写入测试集文件：将每个 query 文件名、documentname、tfidf、tfidf 的排名、bm25、bm25 的排名写入 10152130138Testdata 文件

3. 建模—测试

3.1.根据生成的 10152130138Traindata 文件建立逻辑回归模型：

```
def logreRression(features,flag):

    logit=sm.Logit(flag,features)
    model=logit.fit()
    resR=model.predict(testFeatures)

    return resR
```

(调用逻辑回归的库函数)

3.2.对生成的 10152130138Testdata 文件进行测试，得到每个 doc 对应于相应的 query 的概率，并根据这个概率排序。

```
def outputFile(result):
    i=0
    test=""
    dicOffi={}
    dicTest={}
    f=open('10152130138_丁婉宁.res','w')
    for file in testFile:
        if test==file[0]:
            dicTest[file[1]]=result[i]
        else:
            if i!=0:
                dicOffi[test]=dicTest
                dicTest={}
            test=file[0]
            dicTest[file[1]]=result[i]
        i=i+1
    for key,value in dicOffi.items():
        valueSort =sorted(value.items(),key=lambda d:d[1],reverse=True)
        for key1,value1 in valueSort:
            f.write(str(key)+' '+str(key1)+' '+str(value1)+"\n")
    getEvaluation(dicOffi)
```

3.3.评测

```
def getEvaluation(dicOffi):
    fS=open('./2017_test_qrels/qrel_abs_test.txt')
    for line in fS.readlines():
        lineList=line.split()
        if lineList[0] in resS.keys():
```

```

        if lineList[-1]=='1':
            test=resS[lineList[0]]
            test.append(lineList[-2])
            resS[lineList[0]]=test
    else:
        if lineList[-1]=='0':
            test=[]
            test.append(lineList[-2])
            resS[lineList[0]]=test

MAP=0
for key,value in dicOffi.items():
    valueSort =sorted(value.items(),key=lambda d:d[1],reverse=True)
    t=0
    keyList=valueSort.keys()
    for i in range(len(resS[key])):
        t=t+i/((keyList.index(resS[key][i]))+1)
    MAP=MAP+t/(i+1)
MAP=MAP/(len(dicOffi.keys()))

```

四. 实验结果

得到MAP评价为0.9

10152130138_丁婉宁.res:

```

1 CD010653 3942470 0.499559076088
2 CD010653 3154521 0.490472792803
3 CD010653 1780431 0.49015971475
4 CD010653 6858711 0.490032036383
5 CD010653 6898195 0.48959129546
6 CD010653 3434330 0.489036395528
7 CD010653 20957324 0.488880497623
8 CD010653 328363 0.488297402496
9 CD010653 21048473 0.488169744748
10 CD010653 9000320 0.487586688652
11 CD010653 3446304 0.484588536573
12 CD010653 9754689 0.484148043535
13 CD010653 19847739 0.482854927151
14 CD010653 679137 0.482144670883
15 CD010653 4016481 0.481704323126
16 CD010653 17698323 0.481306935588
17 CD010653 760695 0.480994185171
18 CD010653 8833693 0.480866642481
19 CD010653 9075301 0.480553913101
20 CD010653 517150 0.479574142457

```


五. 实验结论

1. Pointwise方法仅仅使用传统的分类，回归或者Ordinal Regression方法来对给定查询下单个文档的相关度进行建模。这种方法没有考虑到排序的一些特征，比如文档之间的排序结果针对的是给定查询下的文档集合，而Pointwise方法仅仅考虑单个文档的绝对相关度；另外，在排序中，排在最前的几个文档对排序效果的影响非常重要，Pointwise没有考虑这方面的影响。

2. 由于本实验的训练数据的 label 是 1 或者 0，所以使用逻辑回归较好。

在选择特征时，由于如果只有 bm25、tfidf不够合理，所以加上他们的排名更加合理。

六. 实验拓展

LTR的学习方法分为单文档方法（Pointwise）、文档对方法（Pairwise）和文档列表方法（Listwise）三类。Pointwise和Pairwise把排序问题转换成回归、分类或有序分类问题。Listwise把Query下整个搜索结果作为一个训练的实例。3种方法的区别主要体现在损失函数（Loss Function）上。

Pointwise方法仅仅使用传统的分类，回归或者Ordinal Regression方法来对给定查询下单个文档的相关度进行建模。这种方法没有考虑到排序的一些特征，比如文档之间的排序结果针对的是给定查询下的文档集合，而Pointwise方法仅仅考虑单个文档的绝对相关度；另外，在排序中，排在最前的几个文档对排序效果的影响非常重要，Pointwise没有考虑这方面的影响。

相比于Pointwise方法，Pairwise方法通过考虑两两文档之间的相对相关度来进行排序，有一定的进步。但是，Pairwise使用的这种基于两两文档之间相对相关度的损失函数，和真正衡量排序效果的一些指标之间，可能存在很大的不同，有时甚至是负相关，另外，有的Pairwise方法没有考虑到排序结果前几名对整个排序的重要性，也没有考虑不同查询对应的文档集合的大小对查询结果的影响(但是有的Pairwise方法对这些进行了改进，比如IR SVM就是对Ranking SVM针对以上缺点进行改进得到的算法)。

相比于Pointwise和Pairwise方法，Listwise方法直接优化给定查询下，整个文档集合的序列，所以比较好的解决了克服了以上算法的缺陷。Listwise方法中的LambdaMART(是对RankNet和LambdaRank的改进)在Yahoo Learning to Rank Challenge表现出最好的性能。