山东大学计算机科学与技术学院

数据挖掘实验报告

实验题目： Homework 1: VSM and KNN		学号： 201834882
日期：2018.10.30	班级： 18 级专硕班	姓名： 谢升军
Email：1640893020@qq.com		

实验目的：实现 VSM 模型和基于 VSM 模型的 KNN 算法

1.预处理文本数据集，并且得到每个文本的 VSM 表示。

2.实现 KNN 分类器，测试其在 20Newsgroups 上的效果

硬件环境：win10 系统的 8g 内存计算机

软件环境：Anaconda3，Spyder

实验步骤与内容：

一、 整体概述

1.VSM 模型

VSM 概念简单，把对文本内容的处理简化为向量空间中的向量运算，并且它以空间上的相似度表达语义的相似度，直观易懂。当文档被表示为文档空间的向量，就可以通过计算向量之间的相似性来度量文档间的相似性。文本处理中最常用的相似性度量方式是余弦距离。

（1） 文档(Document):泛指一般的文本或者文本中的片断(段落、句群或句子),一般指一篇文章,尽管文档可以是多媒体对象,但是以下讨论中我们只认为是文本对象,本文对文本与文档不

加以区别″。

（2） 项(Term):文本的内容特征常常用它所含有的基本语言单位(字、词、词组或短语等)来表示,这些基本的语言单位被统称为文本的项,即文本可以用项集(Term List)表示为D(T1,T2,,,,Tn)其中是项,1≤k≤n″

（3） 项的权重(TermWeight):对于含有 n 个项的文本D(,……, ,项常常被赋予一定的权重表示他们在文本 D 中的重要程度,即 D=（, ,,, ••••••,)。这时我们说项的权重为(1≤k≤n)。本文的权重用 TF-IDF 计算，计算公式如下：

$$tf(t,d) = \alpha + (1-\alpha)\frac{c(t,d)}{\max_t c(t,d)}$$

$$IDF(t) = \log(\frac{N}{df(t)})$$

$$w(t,d) = TF(t,d) \times IDF(t)$$

本实验计算 TF 公式的 α 取 0。

（4）相似性计算：用两个向量的 cosine 值来衡量相似性。Cosine 值计算公式如下：

$$cosine(d_i, d_j) = \frac{V_{d_i}^T V_{d_j}}{|V_{d_i}|_2 \times |V_{d_j}|_2}$$

2. KNN 算法实现分类

首先将训练数据文档表示为一个个的向量，然后把测试文档表示成向量,判断测试文档分类的 knn 方法是计算测试向量与每个训练向量的 cosine 值，选择 cosine 值最大的 top k 个向量，然后在这 top

k 个训练向量中选择拥有最多向量的类别作为测试向量分类的判定。

二、 具体实现

1. 将文件分为 80%的训练数据和 20%的测试数据并进行预处理

```
def createFiles():        #读取文件进行预处理并写入到新的文件中去
    srcFilesList = listdir(pa)
    for i in range(len(srcFilesList)):
        dataFilesDir = pa + '/' + srcFilesList[i] # 20个文件夹每个的路径
        dataFilesList = listdir(dataFilesDir) # 每个文件夹中每个文件的路径
        for j in range(len(dataFilesList)):
            x = -1
            m = randint(1,10)
            n = randint(1,10)
            if j%10 == n or j%10 == m: #将文件分为80%的训练数据和20%的测试数据
              x = 1                    #标记为1则分为测试数据
            else:
              x = 0                    #标记为0则分为训练数据
            createProcessFile(srcFilesList[i],dataFilesList[j],x)# 调用creat
            print ('%s %s' % (srcFilesList[i],dataFilesList[j]))
```

此方法用于将文件分为 80%的训练数据和 20%的测试数据，在遍历文件时每 10 个文件选择两个随机数文件作为测试文件，其他作为训练文件，选择完后将文件分别写入到新的分为训练和测试文件夹的文件中。

```
def lineProcess(line):          #调用nltk对数据进行预处理
    stopwords = nltk.corpus.stopwords.words('english') #去停用词
    porter = nltk.PorterStemmer()  #词干分析
    splitter = re.compile('[^a-zA-Z]')  #去除非字母字符，形成分隔
    words = [porter.stem(word.lower()) for word in splitter.split(line)\
             if len(word)>0 and\
             word.lower() not in stopwords]
    return words
```

此方法用 nltk 对文字进行预处理，分别进行了去停用词，词干分析，去除非字母字符等操作

```
mathew
mathew
manti
co
uk
subject
univers
violat
separ
church
state
dmn
kepler
```

预处理完后的词语存在新的文件中如上，每个单词一行，方便读取。

2.生成词典同时计算每个单词的 IDF

```
def createdicount(): #计算词频和idf
    n = filecount()  #计算文件数量
    wordMap = {}     #存储词频
    worddf = {}      #存储df
    newWordMap = {}
    fileDir = targettrain
    sampleFilesList = listdir(fileDir)
    for i in range(len(sampleFilesList)):
        sampleFilesDir = fileDir + '/' + sampleFilesList[i]
        sampleList = listdir(sampleFilesDir)
        for j in range(len(sampleList)):
            sampleDir = sampleFilesDir + '/' + sampleList[j]
            temp = open(sampleDir).readlines()
            tempdic = Counter(temp)  #调用counter函数计算文件单词的词频
            for key,value in tempdic.items():
                key = key.strip('\n')   #去除空格
                wordMap[key] = wordMap.get(key,0) + value  #计算词频
                worddf[key] = worddf.get(key,0) + 1        #计算原始df
    #只返回出现次数大于2的单词
    for key, value in wordMap.items():
        if value > 2:
            newWordMap[key] = worddf[key]
    sortedNewWordMap = sorted(newWordMap.items())  #将词典按字母顺序排序
    newworddf = { }
    for i in sortedNewWordMap:
         newworddf[i[0]] = math.log10(n/i[1])  #计算真实tf，
    return newworddf
```

此方法用于生成词典并计算每个单词的词频和 IDF，遍历训练数据文件夹中的文件，将单词和词频存到一个字典中，将单词和 df 存到另外一个字典中。然后根据只选择词频大于 2 的单词放到新的字典中去，得到的字典大小约为 35000 个单词，在可以接受的范围里。最后计算 IDF，根据概述的公式求得。返回存贮 IDF 的字典。

3. 计算每个向量的每个 TF 值

```
def computetf(dic,vocabulary): #dic是个字符串列表，计算df，vocabula
    dicVector = {}
    for i in dic:
        if i in vocabulary:  #只有在词典的单词才计算，不在词典的词语
          if i in dicVector.keys():
            dicVector[i]  = dicVector[i]+1
          else:
            dicVector[i]  = 1
    m = max(dicVector, key=lambda x: dicVector[x]) #求文件中出现词
    w = dicVector[m]
    for key,value in dicVector.items():
        dicVector[key] = value/w                  #tf用文件中出现词频数除
    return  dicVector
```

此方法计算每个向量的 tf 值，dic 参数是每个文件中的单词 list Vocabulary 是词典 list，此方法将只将 dic 中和 vocabulary 的都有的单词存到向量字典 key 值中去，字典的 value 值存真实 tf 值，返回存储此向量的字典。

4. 生成向量 list，并把生成的向量存到文件中去

```
def computevector(inpath,savepath):#生成向量
    idf = createdicount()          #idf 的值 存在字典里
    classfrom = []                 #保存向量类别
    vectorclass = []              #保存向量
    vocabulary =  createvocabulary(idf)#保存词典
    srcFilesList = listdir(inpath)
    for i in range(len(srcFilesList)):
        dataFilesDir = inpath + '/' + srcFilesList[i] # 20个文件夹每个的路径
        dataFilesList = listdir(dataFilesDir) #每个文件夹中每个文件的路径
        for j in range(len(dataFilesList)):
            a = dataFilesDir+'/'+dataFilesList[j]
            b = srcFilesList[i]+'/'+dataFilesList[j]
            classfrom.append(b)
            fr = open(a,'r')
            temp = fr.readlines()
            fr.close()
            for k in range(len(temp)):
              temp[k] = temp[k].strip()
            #print (temp)
            vector = computetf(temp,vocabulary)
            for key,value in vector.items():
                vector[key] = idf[key]*value
            vectorclass.append(vector)
```

此方法前半部分遍历文件夹，生成每个文件的向量，每个向量的值为 idf*tf，此处的 idf 和 tf 分别调用 creatdicount（）方法和 computetf（）生成。将所有向量存到 vectorclass 的 list 中，同时将它的类别存到 classfrom list 中。

```
    for i in range(len(vectorclass)):  #将向量写到文件中去
      tpath = classfrom[i].split('/')
      fpath = savepath + '/' + classfrom[i]
      targetvectordir =  savepath + '/'+ tpath[0]
      if path.exists(targetvectordir)==False:
            mkdir(targetvectordir)
      fw = open(fpath,'w')
      for key,value in vectorclass[i].items():
          fw.write( key +'/'+ str(value)+'\n')
      fw.close()
    print('done')
    return classfrom,vectorclass
```

然后将 vectorclass 中的向量存到文件中去。生成的文件如下：

```
mathew/2.286500824547231
manti/0.789637013575018
co/0.46136052091906016
uk/0.4166699783379443
subject/0.0
univers/0.9299320549218661
violat/0.5995954987767148
separ/1.0752816536761147
```

5.计算 KNN,并计算正确率

```
def knn():
    testvsm ,testvfrom =  creatvector(vectortestpath)
    trainvsm ,trainvfrom =  creatvector(vectortrainpath)
    testmo = [ ]  #存储测试向量的模长
    trainmo = [ ] #存储训练向量的模长
    k = 3         #knn的k的取值
    result = [ ]  #存储识别结果
    accurate = [ ]#存储是否识别正确
    for i in range(len(testvsm)): #计算模长
        tempsum = 0
        for key,value in testvsm[i].items():
            tempsum = tempsum +value*value
        testmo.append(math.sqrt( tempsum ))
    for i in range(len(trainvsm)):
        tempsum = 0
        for key,value in trainvsm[i].items():
            tempsum = tempsum +value**2
        trainmo.append(math.sqrt( tempsum ))
    #print(trainmo)
    for i in range(len(testvsm)):
        topk = []
        classfrom = []
        for j in range(len(trainvsm)):
            tempv = 0#用来计算两个向量的积
            for key,value in testvsm[i].items():
                if key in trainvsm[j].keys():
                    tempv = tempv + value*trainvsm[j][key]
            cosine = tempv/(trainmo[j]*testmo[i])  #计算cos值
            if len(topk)<k :
                topk.append(cosine)
                classfrom.append(trainvfrom[j])
            elif cosine >= min(topk):  #去前k个最大的cosine值
                topk[topk.index(min(topk))] = cosine
                classfrom[topk.index(min(topk))] = trainvfrom[j]
```

将测试数据向量和训练数据向量分别存到 testvsm 和 trainvsm 中

然后遍历 testvsm 中的向量，分别与 trainvsm 中的向量计算 cosine 值，将最大的 k 个值存到 topk list 中，选择里边向量最多的类作为结果存到 result list

```
            classfrom[topk.index(min(topk))] = trainvfrom[j]
        a = Counter(classfrom)
        '''if len(a) >= 3: #看k个备选类里出现最多的前三个类，有一个满足则命中
          res = a.most_common(3)
          if res[0][0] == testvfrom[i] or res[1][0] == testvfrom[i] or res[2][0] == test
            accurate.append(1)
          else:
            accurate.append(0)
        else:
          res = a.most_common(1)
          if res[0][0] == testvfrom[i] :#or res[1][0] == testvfrom[i] :
            accurate.append(1)
          else:
            accurate.append(0)'''
        res = a.most_common(1)  #看k个备选类里出现最多的类，有一个满足则命中
        if res[0][0] == testvfrom[i] :#or res[1][0] == testvfrom[i] :
            accurate.append(1)
        else:
            accurate.append(0)
        result.append(res[0])
    #print (topk)
    #print (classfrom)
s = 0
for i in range(len(accurate)):
    if accurate[i] == 1:
        s = s + 1
f = s/ len(accurate)
print(result)
print(f)
```

然后将 result list 中的结果和 testfrom 中的此向量的真实类别做比较，如果正确则存 1 到 accurate list 中去。最后根据 accurate list 的 1 的数量的比例计算正确率。

三、 实验结果

本实验的词典大小为 35642，当选择 k 为 10 时，此时正确率为

0.8206263840556786

当 k 取 5 时，此时正确率为

```
0.8329642518190447
```

当 k 取 3 时，此时正确率为

```
('talk.politics.guns', 1),
('talk.religion.misc', 3),
('talk.religion.misc', 3),
('talk.religion.misc', 2),
('talk.religion.misc', 3),
('talk.religion.misc', 1),
('soc.religion.christian', 2)
('talk.religion.misc', 3),
('talk.religion.misc', 2)]
0.8389750079088896
```

因为向量用词典存储，只存储文件中有的单词，所以计算的速度还是挺快的，效率也挺高，七八分钟就能在普通计算机上跑完。

结论分析与体会：

本次实验收获巨大，首先是之前没用过 python，借着这次实验的机会学习了 python 这种很实用的工具。在此次实验遇到了好多问题，第一个遇到的问题就是读文件时总是出错，后来在网上查询后的得知 open 函数中有一个参数 errors，因为数据有可能有错误的需要设置为 replace；第二个遇到问题就是向量如何存储是存到矩阵还是存到字典中去，后来经过分析存到矩阵后减少很多计算，所以就采取存到字典中，虽然操作稍微有点麻烦；第三个遇到的问题是在计算 knn 是选择最大的 k 个 cosine 值，我刚开始选择的最小的 k 个 cosine 值，得到的准确率出来是百分之二就有点崩溃了，一度曾怀疑这个算

法的正确性，后来和同学打乒乓球不经意的一句提醒恍然大悟。