推荐系统第二次作业

——实现基于内容的推荐算法

苏晨希

2021 年 5 月 5 日

Contents

1 问题介绍

互联网时代, 人们每天都在接收着海量的信息。我们也常常会遇到这样一个场景：早上起床, 打开手机, 手机上自动推送了几条新闻, 发现有自己感兴趣的新闻, 于是下意识地点进去查看。

本次实验的背景正是这种情况, 数据包括了用户, 他曾经点击过的新闻, 某一页新闻推荐页面和他点击了哪些新闻, 以及每条新闻的具体内容。我们的任务是对于某个用户, 预测他点击某些新闻的概率。评测指标为：(precision@1+precision@2+precision@3)/3(在我们对于预测的概率从大到小排序后)

训练数据的格式如下：

	Uid	Date	History	Impression
0	U48196	11/11/2019 7:56:24 PM	N51591 N23288 N33604 N3908 N16730 N23571 N1673...	N58410-0 N53696-0 N51797-0 N64482-0 N4936-0 N1...
1	U46641	11/14/2019 1:03:08 PM	N44598 N48076 N31248 N11673 N18106 N3128	N20270-0 N34185-0 N38779-1 N45523-0
2	U48973	11/14/2019 9:47:23 AM	N27612 N24569 N51706 N63526 N62058	N23446-0 N12446-0 N58660-0 N38779-0 N6477-0 N3...
3	U15980	11/12/2019 6:14:10 AM	N52518 N16636 N32223 N60485 N43725 N9460 N3654...	N57713-0 N25722-0 N20043-0 N20041-0 N57459-0 N...
4	U48830	11/10/2019 1:42:02 PM	N1570 N9808 N39813 N10059 N30708 N15771 N22469...	N53245-0 N4936-0 N27101-0 N64513-0 N15224-0 N3...
...	...	...	...	...

其中：Uid 表示用户 id,Date 表示记录的时间,History 表示用户点击历史, Impression 表示某推荐页面, 后面标-1 表示点击了, 标-0 表示未点击。

关于新闻的训练数据：

	Nid	Category	SubCategory	Title	Abstract
0	N55528	lifestyle	lifestyleroyals	The Brands Queen Elizabeth, Prince Charles, an...	Shop the notebooks, jackets, and more that the...
1	N19639	health	weightloss	50 Worst Habits For Belly Fat	These seemingly harmless habits are holding yo...
2	N61837	news	newsworld	The Cost of Trump's Aid Freeze in the Trenches...	Lt. Ivan Molchanets peeked over a parapet of s...
3	N53526	health	voices	I Was An NBA Wife. Here's How It Affected My M...	I felt like I was a fraud, and being an NBA wi...
4	N38324	health	medical	How to Get Rid of Skin Tags, According to a De...	They seem harmless, but there's a very good re...
...	...	...	...	...	...

其中：Nid 表示新闻 id,Category、SubCategory、Title、Abstract 分别表示新闻的类别、子类别、标题、摘要, 这四者可以看作是新闻的特征。

测试数据与训练数据几乎一致，仅仅是关于用户的数据中 Impression 未标明是否点击过, 其余均一致, 而我们需要做的就是预测用户点击 Impression 中各条新闻的概率。

2 数据分析

2.1 整体分析

先使用 dataframe.describe() 函数对于数据整体感受一下：

(1) 训练的用户数据:

```
train_data=pd.read_csv("train/train.tsv",sep='\t')
train_data.describe()
```

	Uid	Date	History	Impression
count	10000	10000	10000	10000
unique	8651	9849	8644	9568
top	U56029	11/14/2019 12:41:18 PM	N20263 N6233 N13480 N941 N3508 N56742 N33969 N...	N55689-1 N35729-0
freq	5	3	5	28

训练的用户数据中一共有 10000 条记录, 但是只有 8651 个不同的用户 id, 说明存在同一个用户在不同时间点的记录, 通过查看具体数据也证实了这一点, 从这里我们可以发现时间这一特征也是需要考虑的重要因素。

关于其中涉及到的新闻的总数目, 不能通过 unique 查看, 需要另外书写代码, 将在 2.2 节展示

(2) 测试的用户数据:

```
test_data=pd.read_csv("test/test.tsv",sep='\t')
test_data.describe()
```

	Uid	Date	History	Impression
count	2000	2000	2000	2000
unique	1969	1966	1967	1911
top	U10012	11/15/2019 10:30:08 AM	N53712 N11056 N17933 N7333 N41997 N27777 N1959...	N20036 N36779
freq	2	3	2	30

测试的用户数据中一共有 2000 条记录, 只有 1969 个不同的用户 id, 在进行预测时, 我们需要考虑时间这一特征。

(3) 训练的新闻数据:

```
train_data_news=pd.read_csv("train/train_news.tsv",sep='\t')
train_data_news.describe()
```

	Nid	Category	SubCategory	Title	Abstract
count	51282	51282	51282	51282	48616
unique	51282	17	264	50434	47309
top	N5143	news	newsus	Photos of the Day	What's the weather today? What's the weather f...
freq	1	15774	6564	15	124

训练的新闻数据中一共有 51282 条数据, 有 51292 个不同的新闻 id, 说明没有重复新闻, 每个新闻对应一个信息。数据中一共有 17 种不同的新闻类别,264 种不同的新闻子类别, 50434 种不同的标题和 47309 种不同的摘要。从数量上来看, 如果我们需要使用标题或是摘要作为特征, 我们不能直接使用, 而应该先进行特征的提取。

(4) 测试的新闻数据:

```
test_data_news=pd.read_csv("test/test_news.tsv",sep='\t')
test_data_news.describe()
```

	Nid	Category	SubCategory	Title	Abstract
count	42416	42416	42416	42416	40395
unique	42416	17	257	41823	39470
top	N41409	news	newsus	Look of the Day	What's the weather today? What's the weather f...
freq	1	13043	5335	16	68

测试的新闻数据中一共有 42416 条数据, 有 42416 个不同的新闻 id, 说明没有重复新闻, 每个新闻对应一个信息。数据中一共有 17 种不同的新闻类别,257 种不同的新闻子类别, 41823 种不同的标题和 39470 种不同的摘要。从数量上来看, 如果我们需要使用标题或是摘要作为特征, 我们不能直接使用, 而应该先进行特征的提取。

2.2 用户数据中涉及到的新闻总数目

```
#用于检测出现在data中的所有新闻是否都出现在data_news中
def check(data,data_news,f): #f=1表示是训练数据,f=0表示是测试数据
    newsset=set() #用于记录所有出现在data中的新闻id
    for row in data.itertuples(): #对于data来说,新闻id涉及History和Impression两部分
        History=getattr(row,'History')
        Impression=getattr(row,'Impression')
        #计算History
        L=History.split(' ') #使用split函数分离出各个新闻
        for e in L:
            if e not in newsset:
                newsset.add(e)
        #计算Impression
        L=Impression.split(' ')
        for e in L:
            if f==1:
                news=e[:len(e)-2] #训练数据需要去处最后关于是否点击的标签
            else:
                news=e
            if news not in newsset:
                newsset.add(news)
    print(len(newsset))

    allnews=set() #用于记录所有出现在data_news中的新闻id
    for row in data_news.itertuples():
        news=getattr(row,'Nid')
        if news not in allnews:
            allnews.add(news)

    flag=True
    for e in newsset:
```

```
        if e not in allnews:
            print("Exist news which viewed by the user that not in news.tsv!")
            flag=False
            break
    if flag:
        print("No error!")

print("train:")
check(train_data,train_data_news,1)
print("test:")
check(test_data,test_data_news,0)
```

```
train:
28871
No error!
test:
13587
No error!
```

结果表明, 训练用户数据中一共涉及到 28871 种新闻, 他们都在训练新闻数据中出现过。而测试用户数据中一共涉及到了 13587 种新闻, 他们也都包含在测试新闻数据中。我们获得的关于新闻的数据集比我们所需要的还多出一部分。

2.3 训练数据和测试数据的交叉

我们注意到, 此次实验有一些特别, 对于测试数据来说, 也有它的一部分数据信息, 这就引起了一个问题，即：是否需要把训练数据中的信息加入最终的模型中？还是仅仅使用训练数据得到一些模型参数，而仅仅使用测试数据？

为了得到解答，在 2.3 节会比较所有涉及到的用户, 看看是否有一样的。

```
#用于比较训练用户集中的用户和测试用户集中的用户
def checkuser():
    trainusers=set() #用户集中的用户id
    for row in train_data.itertuples():
        uid=getattr(row,'Uid')
        if uid not in trainusers:
            trainusers.add(uid)

    testusers=set() #测试集中的用户id
    for row in test_data.itertuples():
        uid=getattr(row,'Uid')
        if uid not in testusers:
            testusers.add(uid)
    #计算两者的交集
    unionset=trainusers.intersection(testusers)
    print(len(unionset))
```

```
checkuser()
```

```
58
```

我们发现存在一些用户, 他们既在训练集中又在测试集中, 对于这些用户我们其实可以利用部分他们在训练集中的信息, 使得预测更加可信。

2.4 NAN 值

文本中可能会出现 NAN 值, 所以这里检测一下:

```
print("train:")
data=pd.read_csv("train/train_news.tsv", sep='\t')
data0=data['Category']
print("Category NAN:",data0.isnull().T.any())
data1=data['SubCategory']
print("SubCategory NAN:",data1.isnull().T.any())
data2=data['Title']
print("Title NAN:",data2.isnull().T.any())
data3=data['Abstract']
print("Abstract NAN:",data3.isnull().T.any())

print("test:")
data=pd.read_csv("test/test_news.tsv", sep='\t')
data0=data['Category']
print("Category NAN:",data0.isnull().T.any())
data1=data['SubCategory']
print("SubCategory NAN:",data1.isnull().T.any())
data2=data['Title']
print("Title NAN:",data2.isnull().T.any())
data3=data['Abstract']
print("Abstract NAN:",data3.isnull().T.any())
```

```
train:
Category NAN: False
SubCategory NAN: False
Title NAN: False
Abstract NAN: True
test:
Category NAN: False
SubCategory NAN: False
Title NAN: False
Abstract NAN: True
```

对于训练集和测试集：有且仅有 abstract 存在 NAN 值

3 核心算法

3.1 思路总览

此次实验是实现基于内容的推荐算法。大概的思想是：对于某个用户，他有自己的阅读偏好, 我们可以通过他之前查阅过的新闻, 找出相关的特征, 并推荐具有相似特征的新闻给他。

其实在本次实验中, 我们同样可以利用第一次实验中的协同过滤算法, 用户与用户之间的相似性、新闻与新闻之间的相似性在本次的实验场景中同样说的通, 但是基于本次实验关注的是为内容, 在此实验报告中不考虑协同过滤与基于内容的推荐方法的模型组合, 而仅仅立足在基于内容的推荐算法上。

再说回来, 基于内容的推荐算法的立足点可以看作是一个用户, 因为上文提到了不考虑结合协同过滤, 所以此时的用户与用户之间是相对独立的, 一个用户不需要也不应该关注其他用户的一些行为, 而仅仅关注在自己的身上。

所以，我们现在有什么呢？以训练数据的角度来看：对于某用户, 我们其实有他的两类信息, 一类是他点击过的, 一类是未点击过的。或者说, 我们的每条信息都可以看作是一些 feature 和一个 label。看上去是一个比较典型的分类问题, 不过我们最终需要输出的是概率。

对于分类问题来说, 比较常用的包括：

(1) 决策树、SVM：

这两者都是非线性的, 一般只用于分类, 没有类似于概率的输出。

(2) 对数几率回归

对数几率回归做的在 $y = ax^T + b$ 外面套一层函数, 大于阈值就为 1, 反之为 0。用已有的结果去拟合这个函数。对于对数几率回归来说,y 值可以直接看作是一种概率。但是问题在于：如果使用 LinearRegression 的话，每个用户应该是对应一个不同的模型。(例如：用户 A 的模型对于体育 feature 会返回 1, 对于新闻会返回 2)。这种模型是因用户而异的, 也就是说一个用户如果在训练集中没有出现过的话 (由 2.3 得：存在部分用户同时存在于训练集和测试集中, 但更多的是仅出现在训练集中) 我们其实并没有负样本 (即我们只有用户点击过的新闻), 而这并不能够帮助我们来训练出一个模型。

分析完 (2), 我突然间意识到可能不对, 这并不是一个分类问题。不如接着来看 (3) 的分析：

(3) 基于余弦相似度的模型

这个方法比较朴素, 主要是自己拟定一个函数衡量两个新闻的相似性。再拟定一个函数衡量最终待查询的新闻与用户点击的新闻的相似性以得到概率。

其实对于训练数据来说, 我们还有一部分的内容是关于用户未点击的新闻的; 但是对于测试数据来说, 我们只有极少的一部分数据有这种信息，而大部分并没有。另外，考虑到时间的影响, 训练集中的这部分信息是不是仍然有价值也值得考虑。总的来说，利用训练集中的数据可以认为是一个对于极小部分的改进, 但是大体上并不考虑。

对于这种方法, 我们去拟合的是相似性这件事情，而不同于 (2) 那种本质上把分类看作结果的行为。所以我们不会有用户不同就没有负样本这种情况，而是可以把所有的数据都用于调整衡量两个特征向量的相似性, 以及特征的权重等等。

所以, 我们的过程可以认为是这个样子：

1. 提取特征向量 (1.1-1.3 可选择)

1.1 TD-IDF

1.2 Topic Models

1.3 Word2Vec

我们注意到新闻数据集中给出了这样四个参数：新闻类别、新闻子类别、新闻标题、新闻摘要。很显然这四个就是新闻的四类 feature。新闻类别和新闻子类别通过查看数据, 基本上可以直接使用；而由 2.1 节 (3)(4) 的分析, 新闻标题、新闻摘要还是需要先进行特征的提取, 之后才可以使用。

2. 对于特征向量降维 (可选择)

PCA

3. 计算向量之间的相似性

余弦相似性

(4. 利用相似性进行推荐)

3.2 评测指标

一个值得关注的点是评测指标。如上文所述，我们得到的训练集看上去非常像是一个分类问题，那是不是应该使用分类的准确率作为指标呢？我刚开始也纠结在这个问题上。随手写的代码分类的准确率也达到了 70%+, 这给了我一种很不安的感觉。仔细观察训练集会发现，我们对于 Impression 进行预测, 预测的结果中大部分都是未点击的, 也就是说即使你全部预测为 0, 也可以得到一个非常不错的准确率, 而这显然不是我们所要的, 这表明了使用分类的准确率作为评测指标并不能很好的描述这个问题。

结合评测指标为 precision, 我想到一个更好的评测指标是：假设某用户一共点击过 x 篇文章, 就选择我预测的准确率 top-x 的文章, 求这两者的交集大小/x。

3.3 特征提取

对于文本进行特征提取的方法很多, 本实验报告主要使用的 TF-IDF 这一方法。

TF-IDF 主要关注两个部分：

(1)TF 用于关注一个词语在一篇文章中出现的频率。

TF(i,j) 表示词语 i 在文章 j 中的出现频率。

(2)IDF 主要用于减小在多篇文章中都多次出现的词语的权重。

IDF(i) 表示某词语 i 在所有文章中出现的频率的倒数。

TF-IDF(i,j) 表示词语 i 在文章 j 中的权重。

TF-IDF(i,j)=TF(i,j)*IDF(i)

3.3.1 实现 TF-IDF 的流程

实现 TF-IDF 的流程可以认为是：

(1) 对于每篇文章 j, 计算每个词语 i 的出现频率; 得到 TF(i,j)

(2) 对于所有的文章, 统计某个词语 i 在所有文章中出现的的频率的倒数; 得到 IDF(i)

(3) 对于每篇文章 j, 每个词语 i, 计算得到 TF-IDF(i,j)。

由 2.4 节的分析,Abstract 中存在 NAN 值, 如果需要使用需要预处理。

另外, 由于 sklearn 的 *feature_extraction.text* 模块中提供了相关工具, 所以就不用自己实现了, 具体见 3.7 节

3.4 衡量两个新闻的相似性

一个新闻可以看作是 4 个 featurevector 组成, 分别表示 category,subcategory,title,abstract。两个新闻之间的相似性也就是多个 feature 加权平均：

$$sim(News1, News2) = \sum_{i=1}^{4} weight[i] * sim(feature[news1][i], feature[news2][i])$$

衡量两个特征向量的相似性常用的是余弦相似性。即对于两个 vector 来说：

$$\cos(\overrightarrow{a}, \overrightarrow{b}) = \frac{\sum a_i * b_i}{\sqrt{\sum a_i^2} * \sqrt{\sum b_i^2}}$$

sklearn 中同样提供了 *cosine_similarity* 函数, 相比于自己写的函数会更快一点, 所以这里使用 sklearn 自带的函数。

3.5 计算用户点击某新闻的概率

对于用户所有历史浏览过的新闻, 依次和待预测的新闻计算相似度求平均值。

$$sim(user, news) = \frac{\sum_{e \in History} sim(news, e)}{|History|}$$

由 2.1 节可以了解到, 同样 id 的用户有多条, 所以更准确一点, 这里的相似度应该为 $sim(history, news)$

3.6 整体伪代码

Algorithm 1 Content-based Recommendation System

Require: data,datanews

```
load-data,preprocess
calculate TF-IDF for Title and Abstract to generate feature3-4
Use PCA to reduce feature-vectors for feature1-4(if needed)
for each line do
    for each news n waiting for prediction  do
        calculate sim(history,n)
    end for
end for
```

3.7 具体代码注解

由于电脑内存等问题, 本来是打算写成一整个类的, 但是没办法运行, 所以从 3.7.2 部分开始拆成一个个外面的函数。

3.7.1 load-data,preprocess

```
class ContentBasedRecommendation:
    def __init__(self):
        self.weighted=[0.25,0.25,0.25,0.25] #weight of each feature
        self.category=dict() #{category:index}
        self.subcategory=dict() #{subcategory:index}
        self.categorynum=0 #category的总数
        self.subcategorynum=0 #subcategory的总数
        self.data=None     #用户数据
        self.datanews=None #新闻数据
        self.news=dict()   #{Nid:(category,subcategory,title,abstract)}
        self.Nidset=set()  #记录新闻id的集合
        self.NidDict=dict() #{Nid:index}
    def LoadData(self,path1,path2):
        #数据导入,由于是tsv格式,所以使用pd.read_csv(sep='\t')
        self.data=pd.read_csv(path1,sep='\t')
        self.datanews=pd.read_csv(path2,sep='\t')
        self.datanews['Abstract'].fillna("is",inplace=True) #Abstract存在NAN值,这里用is填充
    def preprocess(self):
        #部分预处理函数,主要用于产生__init__中提及的结构
        for row in self.datanews.itertuples():#遍历新闻中的每一行
            Nid=getattr(row,'Nid')
            ca=getattr(row,'Category')
            subca=getattr(row,'SubCategory')
            tit=getattr(row,'Title')
            ab=getattr(row,'Abstract')
            if Nid not in self.Nidset:
                self.Nidset.add(Nid) #如果没有在新闻集合中,加入新闻集合
            if Nid not in self.news: #如果不在新闻数据字典中, 加入字典
                self.news[Nid]=[ca,subca,tit,ab]
        #用于产生新闻id对应的编号字典,方便后面的操作
        num=0
        for e in self.Nidset:
            if e not in self.NidDict:
                self.NidDict[e]=num
                num+=1

        for row in self.data.itertuples():#遍历用户中的每一行
            his=getattr(row,'History')
            his=his.split(' ') #对于history先用空格分词
            imp=getattr(row,'Impression')
            imp=imp.split(' ') #对于impression先用空格分词

        #计算category并对应编号
        num=0
```

```
        for row in self.datanews.itertuples():
            c=getattr(row,'Category')
            if c not in self.category:
                num+=1
                self.category[c]=num
        self.categorynum=num
        #计算subcategory并对应编号
        num=0
        for row in self.datanews.itertuples():
            c=getattr(row,'SubCategory')
            if c not in self.subcategory:
                num+=1
                self.subcategory[c]=num
        self.subcategorynum=num
    def getNewsfeature1(self,Nid)
    def getNewsfeature2(self,Nid)
    def main(self,path1,path2):
        #主程序,调用了load-data和preprocess
        self.LoadData(path1,path2)
        self.preprocess()
```

3.7.2 TF-IDF for Title and Abstract

以 title 为例：abstract 一样

```
        corpus=[] #记录所有新闻对应的feature3(title),用于PCA
        for e in cbr.Nidset:
            corpus.append(cbr.news[e][2])#self.news[e]=(category,subcategory,title,abstract)

        vectorizer=CountVectorizer()#该类会将文本中的词语转换为词频矩阵,
                                    #矩阵元素a[i][j] 表示j词在i类文本下的词频
        transformer=TfidfTransformer()#该类会统计每个词语的tf-idf权值
        tfidf=transformer.fit_transform(vectorizer.fit_transform(corpus))#第一个fit_transform是计算tf-idf,
                                                                         #第二个fit_transform是将文本转为词频矩阵
        word=vectorizer.get_feature_names()#获取词袋模型中的所有词语
        weight=tfidf.toarray()#将tf-idf矩阵抽取出来，weight[i][j]表示j词在i类文本中的tf-idf权重
```

我们已知 tf-idf 矩阵的横坐标是所有词语的总数量, 这个数量级太大了。另外对于一篇文章, 我们需要关注的 feature 不需要很多, 这里可以认为是 tf-idf 前 3 的词语。因此我们对于这些词语再提取成一个新的词典, 重新产生 feature

```
        wordset=set() #记录提取top-3之后的词典
        for i in range(len(weight)):
            #记录一篇文章中的词语和词语的tf-idf值
            L=[]
            for j in range(len(word)):
```

```
            L.append((word[j],weight[i][j]))
        L.sort(key= lambda k:-k[1])
        #排序以后取前3
        for ii in range(min(len(L),3)):
            if L[ii][0] not in wordset:
                wordset.add(L[ii][0])

    featureL=[] #记录重新提取后所有新闻的feature3,用于PCA
    wordnum=len(wordset)
    #为新词典中的词语编号
    worddict=dict()
    num=0
    for e in wordset:
        worddict[e]=num
        num+=1
    for i in range(len(weight)):
        #遍历每篇文章,重新产生feature
        #feature形如[0,0,0.5,0.32,0,0,12...]
        #为0表示没有这个词语,非0表示tf-idf值
        L=[0 for ii in range(wordnum)]
        for j in range(len(word)):
            if word[j] in wordset:
                L[worddict[word[j]]]=weight[i][j]
        featureL.append(L)
```

3.7.3 generate other features

```
class ContentBasedRecommendation:
    def getNewsfeature1(self,Nid):
        #产生Nid的category所对应的feature
        #feature形如[0,0,0,0,1,0,0,...,0],维度是category的总数
        #如果是某个category, 对应的那一位为1
        content=self.news[Nid] #(category,subcategory,title,abstract)
        category=self.category[content[0]]
        feature1=[0 for i in range(self.categorynum)]
        feature1[category-1]=1 #由于categorydict从1开始编号,所以-1
        return feature1
    def getNewsfeature2(self,Nid):
        #产生Nid的subcategory所对应的feature
        #feature形如[0,1,0,0,0,0,0,...,0],维度是subcategory的总数
        #如果是某个subcategory, 对应的那一位为1
        content=self.news[Nid]#(category,subcategory,title,abstract)
        subcategory=self.subcategory[content[1]]
        feature2=[0 for i in range(self.subcategorynum)]
        feature2[subcategory-1]=1#由于subcategorydict从1开始编号,所以-1
```

```
        return feature2

    #外部书写
    feature2List=[] #记录所有新闻的feature2,用于PCA
    for e in cbr.Nidset:
        #遍历新闻集合,获取feature2,存入list
        feature2=cbr.getNewsfeature2(e)
        feature2List.append(feature2)
    feature1List=[]#记录所有新闻的feature1,用于PCA
    for e in cbr.Nidset:
        #遍历新闻集合,获取feature1,存入list
        feature1=cbr.getNewsfeature1(e)
        feature1List.append(feature1)
```

3.7.4 PCA

```
    pca4=PCA(n_components=20)#对于abstract,设定降维后剩下20维
    f4PCA=pca4.fit_transform(feature4)

    pca3=PCA(n_components=20)#对于title,设定降维后剩下20维
    f3PCA=pca3.fit_transform(featureL)

    pca2=PCA(n_components=20)#对于subcategory,设定降维后剩下20维
    f2PCA=pca2.fit_transform(feature2List)

    pca1=PCA(n_components=5)#对于category,设定降维后剩下5维
    f1PCA=pca1.fit_transform(feature1List)
```

3.7.5 train

```
def getNewsfeatures(Nid):
    #input Nid,return [feature1,feature2,feature3,feature4] of a news
    x=cbr.NidDict[Nid] #获取Nid对应的编号
    feature1=f1PCA[x]
    feature2=f2PCA[x]
    feature3=f3PCA[x]
    feature4=f4PCA[x]
    return [feature1,feature2,feature3,feature4]
def cosine(x,y):
    #计算余弦相似性
    if (len(x)==0 and len(y)==0):
        return 0
    L=[x,y]
    return cosine_similarity(L)[0][1]
```

```
def predict(history,news):
    #计算用户对于某新闻的预测
    ans=0
    for n in history: #遍历所有历史点击的新闻,求平均
        fea1=getNewsfeatures(n)
        fea2=getNewsfeatures(news)
        for i in range(4): #两个新闻之间的相似性为多个feature加权平均
            ans=ans+weighted[i]*cosine(fea1[i],fea2[i])
    return ans/len(history)

#外部
tot=0 #记录sum(precision)
totnum=0 #记录总共测试了多少个用户
for row in cbr.data.itertuples(): #遍历所有数据
    hisori = getattr(row, 'History') # 原先的history
    his = hisori.split(' ') # 对于history先用空格分词
    impori = getattr(row, 'Impression') # 原先的impression
    imp = impori.split(' ') # 对于impression先用空格分词
    L=[]      #记录预测结果
    clicked=[] #记录用户点击了那些新闻

    for n in imp: #遍历impression,n形如N13543-0/1
        news=n[:len(n)-2] #获取新闻id
        label=n[len(n)-1] #获取标签("1"表示点击过)
        if (label=="1"):
            clicked.append(news) #如果点击,加入clicked
        predictans=predict(his,news)
        L.append((news,predictans)) #将新闻和预测结果压入L
    #排序,找出top-k
    L.sort(key= lambda k:-k[1])
    mypredict=L[:len(clicked)]

    oriset=set()
    myset=set()
    for each in clicked:
        oriset.add(each)
    for each in mypredict: #each:(news,predict_sim)
        myset.add(each[0])
    #计算预测的和实际的重合个数/点击的总数
    precision=len(oriset&myset)/len(clicked)
    totnum+=1
    tot+=precision
    print(tot/totnum)
```

4 训练结果分析及参数调整

由于电脑性能受限, 训练集为从 train 中随机划分出来的 1000 个用户以及所有里面涉及到的新闻。

4.1 (PCA and top-3 select) or not

虽然设计算法的时候认为 PCA 和对于每篇文章仅选择前 3 关键词可能会有用, 但仍然需要经过测试：(feature weight 取值相同)

Method	ans
PCA and top-3 select	0.156
PCA	0.160
None	0.185

发现什么都不加的效果最好。

分析原因：认为 PCA 是无监督学习, 学习的是输入的维度之间的相关性, 将原来线性无关的列转换为线性相关, 在减少维度的同时保存其原始特征。对于 PCA 来说, 并不能够和结果产生关联, 反而有可能在压缩下使得数据的特征不是那么的清晰。

4.2 feature weight

由于计算速度较慢, 这里仅尝试了几种情况：

(这里的 weight 加起来不等于 1, 仅表示各个 feature 之间的比例关系)

Weight	ans
$[0.3, 0.3, 0.3, 0.3]$	0.185
$[0.3, 0.3, 0.3, 0.2]$	0.188
$[0.3, 0, 3, 0.2, 0.3]$	0.181
$[0.3, 0.2, 0.3, 0.3]$	0.183
$[0.2, 0.3, 0.3, 0.3]$	0.186
$[0.2, 0.3, 0.3, 0.2]$	0.189

在我尝试的几种可能的权重组合中,$[0.2, 0.3, 0.3, 0.2]$ 表现的最好。认为对于 Category 来说, 只有 17 种不同的可能 (2.1 节), 并不能很好的描述某个用户喜欢的新闻的特征; 对于 abstract 来说, 考虑用户的行为, 一般来说也就大概的看看标题就选择点进去或者跳过, 相比之下, 文字长度更大的 abstract 影响力不是很大。

5 预测

对于预测程序, 需要修改的部分不是很多。主要是以下两块：

(1) 为了输出 tsv 文件, 需要记录原先的 Date、History 和 Impression

(2) 对于训练的主要部分：把 clicked 等计算评测指标的部分删除; 另外对于训练集, Impression 后面没有标签, 所以不用再划分新闻和标签。

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity

class ContentBasedRecommendation:
    def __init__(self):
        self.weighted = [0.25, 0.25, 0.25, 0.25] # weight of each feature
        self.category = dict() # {category:index}
        self.subcategory = dict() # {subcategory:index}
        self.categorynum = 0 # category的总数
        self.subcategorynum = 0 # subcategory的总数
        self.data = None # 用户数据
        self.datanews = None # 新闻数据
        self.news = dict() # {Nid:(category,subcategory,title,abstract)}
        self.Nidset = set() # 记录新闻id的集合
        self.NidDict = dict() # {Nid:index}

    def LoadData(self, path1, path2):
        # 数据导入,由于是tsv格式,所以使用pd.read_csv(sep='\t')
        self.data = pd.read_csv(path1, sep='\t')
        self.datanews = pd.read_csv(path2, sep='\t')
        self.datanews['Abstract'].fillna("is", inplace=True) #处理NAN值

    def preprocess(self):
        # 部分预处理函数,主要用于产生__init__中提及的结构
        for row in self.datanews.itertuples(): # 遍历新闻中的每一行
            Nid = getattr(row, 'Nid')
            ca = getattr(row, 'Category')
            subca = getattr(row, 'SubCategory')
            tit = getattr(row, 'Title')
            ab = getattr(row, 'Abstract')
            if Nid not in self.Nidset:
                self.Nidset.add(Nid) # 如果没有在新闻集合中,加入新闻集合
            if Nid not in self.news: # 如果不在新闻数据字典中, 加入字典
                self.news[Nid] = [ca, subca, tit, ab]
```

```
        # 用于产生新闻id对应的编号字典,方便后面的操作
        num = 0
        for e in self.Nidset:
            if e not in self.NidDict:
                self.NidDict[e] = num
                num += 1

        for row in self.data.itertuples(): # 遍历用户中的每一行
            Uid = getattr(row, 'Uid')
            date = getattr(row, 'Date')
            hisori = getattr(row, 'History') # 原先的history
            his = hisori.split(' ') # 对于history先用空格分词
            impori = getattr(row, 'Impression') # 原先的impression
            imp = impori.split(' ') # 对于impression先用空格分词

        # 计算category并对应编号
        num = 0
        for row in self.datanews.itertuples():
            c = getattr(row, 'Category')
            if c not in self.category:
                num += 1
                self.category[c] = num
        self.categorynum = num
        # 计算subcategory并对应编号
        num = 0
        for row in self.datanews.itertuples():
            c = getattr(row, 'SubCategory')
            if c not in self.subcategory:
                num += 1
                self.subcategory[c] = num
        self.subcategorynum = num

    def getNewsfeature1(self, Nid):
        # 产生Nid的category所对应的feature
        # feature形如[0,0,0,0,1,0,0,...,0],维度是category的总数
        # 如果是某个category，对应的那一位为1
        content = self.news[Nid] # (category,subcategory,title,abstract)
        category = self.category[content[0]]
        feature1 = [0 for i in range(self.categorynum)]
        feature1[category - 1] = 1 # 由于categorydict从1开始编号,所以-1
        return feature1

    def getNewsfeature2(self, Nid):
        # 产生Nid的subcategory所对应的feature
        # feature形如[0,1,0,0,0,0,0,...,0],维度是subcategory的总数
        # 如果是某个subcategory，对应的那一位为1
```

```
        content = self.news[Nid] # (category,subcategory,title,abstract)
        subcategory = self.subcategory[content[1]]
        feature2 = [0 for i in range(self.subcategorynum)]
        feature2[subcategory - 1] = 1 # 由于subcategorydict从1开始编号,所以-1
        return feature2

    def main(self, path1, path2):
        # 主程序,调用了load-data和preprocess
        self.LoadData(path1, path2)
        self.preprocess()


cbr = ContentBasedRecommendation()
cbr.main("test/test.tsv", "test/test_news.tsv")
# cbr.main("splittrain.tsv","splitnews.tsv")
print("initialize done")
corpus = [] # 记录所有新闻对应的feature3(title),用于PCA
for e in cbr.Nidset:
    corpus.append(cbr.news[e][2]) # self.news[e]=(category,subcategory,title,abstract)

vectorizer = CountVectorizer() # 该类会将文本中的词语转换为词频矩阵，矩阵元素a[i][j]
    表示j词在i类文本下的词频
transformer = TfidfTransformer() # 该类会统计每个词语的tf-idf权值
tfidf = transformer.fit_transform(
    vectorizer.fit_transform(corpus)) #
        第一个fit_transform是计算tf-idf，第二个fit_transform是将文本转为词频矩阵
word = vectorizer.get_feature_names() # 获取词袋模型中的所有词语
weight = tfidf.toarray() # 将tf-idf矩阵抽取出来，weight[i][j]表示j词在i类文本中的tf-idf权重
print("tf-idf calculate done")
print(weight.shape)
f3PCA = weight
# pca3=PCA(n_components=20)#对于title,设定降维后剩下20维
# f3PCA=pca3.fit_transform(weight)


corpus4 = [] # 记录所有新闻对应的feature4(abstract),用于PCA
for e in cbr.Nidset:
    corpus4.append(cbr.news[e][3]) # self.news[e]=(category,subcategory,title,abstract)

vectorizer = CountVectorizer() # 该类会将文本中的词语转换为词频矩阵，矩阵元素a[i][j]
    表示j词在i类文本下的词频
transformer = TfidfTransformer() # 该类会统计每个词语的tf-idf权值
tfidf = transformer.fit_transform(
    vectorizer.fit_transform(corpus4)) #
        第一个fit_transform是计算tf-idf，第二个fit_transform是将文本转为词频矩阵
word = vectorizer.get_feature_names() # 获取词袋模型中的所有词语
```

```
weight = tfidf.toarray() # 将tf-idf矩阵抽取出来，weight[i][j]表示j词在i类文本中的tf-idf权重
print("tf-idf calculate done")
print(weight.shape)
f4PCA = weight
# pca4=PCA(n_components=20)#对于title,设定降维后剩下20维
# f4PCA=pca4.fit_transform(weight)


feature2List = [] # 记录所有新闻的feature2,用于PCA
for e in cbr.Nidset:
    # 遍历新闻集合,获取feature2,存入list
    feature2 = cbr.getNewsfeature2(e)
    feature2List.append(feature2)
feature1List = [] # 记录所有新闻的feature1,用于PCA
for e in cbr.Nidset:
    # 遍历新闻集合,获取feature1,存入list
    feature1 = cbr.getNewsfeature1(e)
    feature1List.append(feature1)
# pca2=PCA(n_components=20)#对于subcategory,设定降维后剩下20维
# f2PCA=pca2.fit_transform(feature2List)
# pca1=PCA(n_components=5)#对于category,设定降维后剩下5维
# f1PCA=pca1.fit_transform(feature1List)
f2PCA = feature2List
f1PCA = feature1List


# print("PCA done")
def getNewsfeatures(Nid):
    # input Nid,return [feature1,feature2,feature3,feature4] of a news
    x = cbr.NidDict[Nid] # 获取Nid对应的编号
    feature1 = f1PCA[x]
    feature2 = f2PCA[x]
    feature3 = f3PCA[x]
    feature4 = f4PCA[x]
    return [feature1, feature2, feature3, feature4]


def cosine(x, y):
    # 计算余弦相似性
    if (len(x) == 0 and len(y) == 0):
        return 0
    L = [x, y]
    return cosine_similarity(L)[0][1]


def predict(history, news):
```

```
    # 计算用户对于某新闻的预测
    ans = 0
    for n in history: # 遍历所有历史点击的新闻,求平均
        fea1 = getNewsfeatures(n)
        fea2 = getNewsfeatures(news)
        for i in range(4): # 两个新闻之间的相似性为多个feature加权平均
            ans = ans + weighted[i] * cosine(fea1[i], fea2[i])
    return ans / len(history)


print("Start to predict:")
weighted = [0.2, 0.3, 0.3, 0.2]
LUid = []
Lorihis = []
Loriimp = []
Ldate = []
Lpredict = []
for row in cbr.data.itertuples(): # 遍历所有行
    Uid = getattr(row, 'Uid')
    date = getattr(row, 'Date')
    hisori = getattr(row, 'History') # 原先的history
    his = hisori.split(' ') # 对于history先用空格分词
    impori = getattr(row, 'Impression') # 原先的impression
    imp = impori.split(' ') # 对于impression先用空格分词

    LUid.append(Uid)
    Lorihis.append(hisori)
    Loriimp.append(impori)
    Ldate.append(date)
    L = [] # 记录预测结果
    for news in imp: # 遍历impression,news形如N13543
        predictans = predict(his, news)
        L.append(predictans) # 将预测结果压入L
    Lpredict.append(L)
df = pd.DataFrame({'Uid': LUid, 'Date': Ldate, 'History': Lorihis, 'Impression': Loriimp,
     'Predict': Lpredict})
df.to_csv("test.tsv", sep='\t', index=False)
```

6 总结与反思

总的来说, 感觉这一次的作业挺难的。对比第一次作业的难度, 个人觉得提升的不少。

困难 首先从下面几个方面讨论一下我遇到的一些困难:

1、评测指标

看到训练数据的第一眼, 我就觉得这是一个分类问题。的确很像, 看 impression 里面还有 0 和 1 的标签, 再加上输出概率, 让我不由自主的就往对数几率回归的方向去想起了。

当然这个想法有很大的问题, 正如在 3.1、3.2 节中探讨的：

第一点是：以训练集的角度来看, 我们并没有负样本。(3.1)

第二点是：以评测指标的角度来看, 分类的准确率并不能很好的描述这个问题。(3.2)

使用更为直观的相似性的方法和 top-k 的覆盖率更为的恰当。

2、userid 不唯一对应

我的第一版程序运行的很慢, 大概 1min 能运行 train.csv 的一行。这使得我把很多可以预处理的部分都提前到了初始化, 并使用对应的数据结构记录下来以提高效率。

很自然的, 我定义了 self.userdata[Uid], 尝试通过 Uid 作为索引来获得对应的哪一行用户数据, 并由此写完了我的所有代码。

但是在生成最终结果的时候, 我意外的发现并没有 2000 行, 这才意识到在 2.1(2) 分析中, 我们已经得出了存在多个同一用户 id 的情况, 并改写了一定部分的代码。充分体现了数据分析, 以及牢记数据分析的结论的意义。

3、设备受限

在数据较大的情况下，我受到了设备的很大影响。

首先就是第 4 节已经提及的划分了 1000 个数据作为训练集。并且在这种情况下, 运行速度也不尽如人意, 最后调参的时候仅仅选择了几组参数进行尝试而没有像前一次实验那样遍历很多可能性。

另一个很典型的例子是关于 tf-idf 的。tf-idf 的两个维度分别是新闻的个数和词语的个数。

我尝试运行原训练集中仅 title 相关的 tf-idf, 便已经收到了 Jupyter Notebook 关于内存的错误提示, 后来将其从函数中提到最外层运行了起来。但是在之后尝试结合 PCA 和仅选择 top-k keywords 的方法的时候, 内存还是不够, 为了节省内存, 我将范围缩小到了所有用户数据中出现过的新闻, 进行了更多的尝试。

思考 第二部分是一些思考：

1、关于词语/词典的范围问题

在上一部分中提到了因为内存的问题, 我缩小了 tf-idf 中整个新闻集的大小。其实在刚刚计算过原训练集中的 tf-idf 之后, 我就开始思考范围这件事情。使用所有新闻作为范围, 或是所有在用户数据中出现的新闻, 抑或是仅在某个用户中出现的新闻？

我自己的理解是：这些范围都不能算是错, 但是范围越大, 对于里面的词语的独特性的表述就越有道理。

2、关于同时出现在训练集和测试集的用户，和同用户不同时间不同记录

这样两种情况我认为可以说是内存推荐中的小意外。它们对于进行推荐都提供了一些额外的 feature, 但是在本次数据集来看, 所占的比例的确非常小, 以至于难以判断加上这些 feature 造成的效果是

否更好。但是对于混合型的推荐系统的构造仍然带来了一定的意义。

3、关于 NAN 值

在文本数据中难免会存在 NAN 值。在本次实验报告中, 采取的做法为将所有 NAN 值填充为 is, 即某个出现概率比较大的词语。但是仔细想想, 对于 NAN 值来说, 分类单独处理似乎更为恰当？对于计算相似度的某一方含有 NAN 值的情况下, 只使用前三种特征 (即去除 NAN 值的那个特征) 想来更为准确一点。

疑惑 最后是一点疑惑吧。

1、与本题相关

写这道题的时候, 一个我比较困扰的点就是 baseline。对于第一次作业来说, 想到物品的平均打分等等是一个很自然的想法, 但是这道题的 baseline 是什么呢？或许是只采用 category 和 subcategory 吗？

2、参数调节

我们在写代码的时候, 常常会遇到一些含有参数的情况。那么到底选择什么样的参数就会成为一个很大的问题。在目前来说, 感觉参数的设置对于整体结果没有特别大的影响, 但是也不难推测：如果延申到神经网络方面, 参数的设计有可能会导致过拟合等问题。

我自己一直以来都是暴力枚举每个可能的取值, 在训练集上使用自己的指标进行比较, 相对来说感觉效率很低。在本节“困难”一段中提及的运算速度不够快, 导致没有办法遍历所有可能性的情况, 其实并不罕见。当模型复杂起来之后, 动辄几十个参数本来也没有办法暴力枚举。

另外一点是, 以本题为例, 在 4.2 节我得到了几个相对来说结果略好一点的组合, 但是却发现在几个划分不同的数据集上它们的表现也不相同, 这使得我其实并不知道应该选择哪一个 (虽然差异很小)。所以可能希望能够在讲解的时候, 了解到一些关于参数调节的知识吧。

7 附录

7.1 提交文件说明

code/train.py 一份可以用于查看训练结果的代码, 需要修改数据路径。在安装了 7.2 节中所提及的所有库后可通过命令 **python train.py** 运行。(注意：数据的 path 需要按照目录结构设定)

code/run.py 一份用于生成最终的预测结果的代码, 需要修改数据路径。在安装了 7.2 节中所提及的所有库后可通过命令 **python run.py** 运行。(注意：数据的 path 需要按照目录结构设定)

test.tsv 预测结果

10185102231.pdf 一份实验报告

7.2 运行环境说明

本次实验使用 python 语言编写，使用了 python 中的 pandas、scikit-learn。需要安装的包和具体版本如下：

Package	Version
Python	3.8.5
pandas	1.1.3
scikit-learn	0.23.2