智能推荐系统第二次编程作业

——Content-based Recommendation

学号：10185102142

姓名：李泽浩

指导老师：张伟

项目名称：Content-based Recommendation

时间：2021年5月5日

目录

1、基于内容的推荐算法简介

1.1算法详解

CB是最早被使用的推荐算法，它的思想非常简单：根据用户过去喜欢的物品（本文统称为 item），为用户推荐和他过去喜欢的物品相似的物品。而关键就在于这里的物品相似性的度量，这才是算法运用过程中的核心。 CB最早主要是应用在信息检索系统当中，所以很多信息检索及信息过滤里的方法都能用于CB中。

举个简单的例子：在京东上购物的小伙伴们应该都知道，每当你进入任何一个物品页面的时候都会有一个“猜你喜欢”的栏目，这时候他就会根据你经常购买的物品给你推荐相似的物品。例如对我来说：我经常购买互联网类书籍，所以它就会给我推荐类似的书籍。

CB的过程一般包括以下三步：

- 物品表示（Item Representation）：为每个item抽取出一些特征（也就是item的content了）来表示此item；
- 特征学习（Profile Learning）：利用一个用户过去喜欢（及不喜欢）的item的特征数据，来学习出此用户的喜好特征（profile）
- 生成推荐列表（Recommendation Generation）：通过比较上一步得到的用户profile与候选item的特征，为此用户推荐一组相关性最大的item。

举个例子说明前面的三个步骤。随着今日头条的崛起，基于内容的文本推荐就盛行起来。在这种应用中一个item就是一篇文章。

第一步，我们首先要从文章内容中抽取出代表它们的属性。常用的方法就是利用出现在一篇文章中词来代表这篇文章，而每个词对应的权重往往使用信息检索中的tf-idf来计算。利用这种方法，一篇抽象的文章就可以使用具体的一个向量来表示了。

第二步，根据用户过去喜欢什么文章来产生刻画此用户喜好的特征向量了，最简单的方法可以把用户所有喜欢的文章对应的向量的平均值作为此用户的特征向量。比如我经常在今日头条阅读技术科技相关的文章，那么今日头条的算法可能会把我的Profile中的：“互联网”、“大数据”、“机器学习”、“数据挖掘”等关键词的权重设置的比较大。

这样，当我登录头条客户端的时候，他获取到我的用户Profile后，利用CB算法将我的个人Profile与文章Item的Profile的相似度（相似度的衡量可以用余弦相似度Cosine）进行计算，然后按相似度大小取最大的前N个篇文章作为推荐结果返回给我的推荐列表中。

1.2物品表示

真实应用中的item往往都会有一些可以描述它的属性。这些属性通常可以分为两种：结构化的（structured）属性与非结构化的（unstructured）属性。所谓结构化的属性就是这个属性的意义比较明确，其取值限定在某个范围；而非结构化的属性往往其意义不太明确，取值也没什么限制，不好直接使用。比如在交友网站上，item就是人，一个item会有结构化属性如身高、学历、籍贯等，也会有非结构化属性（如item自己写的交友宣言，博客内容等等）。对于结构化数据，我们自然可以拿来就用；但对于非结构化数据（如文章），我们往往要先把它转化为结构化数据后才能在模型里加以使用。真实场景中碰到最多的非结构化数据可能就是文章了（如个性化阅读中）。

1.3特征学习的主要方法

假设用户u已经对一些item给出了他的喜好判断，喜欢其中的一部分item，不喜欢其中的另一部分。那么，这一步要做的就是通过用户u过去的这些喜好判断，为他产生一个模型。有了这个模型，我们就可以根据此模型来判断用户u是否会喜欢一个新的item。所以，我们要解决的是一个典型的有监督分类问题，理论上机器学习里的分类算法都可以照搬进这里。

主要方法有：

- Rocchio算法
- 最近邻方法（简称KNN）

1.4优缺点

CB的优点：

- 用户之间的独立性（User Independence）：既然每个用户的profile都是依据他本身对item的喜好获得的，自然就与他人的行为无关。而CF刚好相反，CF需要利用很多其他人的数据。CB的这种用户独立性带来的一个显著好处是别人不管对item如何作弊（比如利用多个账号把某个产品的排名刷上去）都不会影响到自己。
- 好的可解释性（Transparency）：如果需要向用户解释为什么推荐了这些产品给他，你只要告诉他这些产品有某某属性，这些属性跟你的品味很匹配等等。
- 新的item可以立刻得到推荐（New Item Problem）：只要一个新item加进item库，它就马上可以被推荐，被推荐的机会和老的item是一致的。而CF对于新item就很无奈，只有当此新item被某些用户喜欢过（或打过分），它才可能被推荐给其他用户。所以，如果一个纯CF的推荐系统，新加进来的item就永远不会被推荐:(。

CB的缺点：

- item的特征抽取一般很难（Limited Content Analysis）：如果系统中的item是文档（如个性化阅读中），那么我们现在可以比较容易地使用信息检索里的方法来“比较精确地”抽取出item的特征。但很多情况下我们很难从item中抽取出准确刻画item的特征，比如电影推荐中item是电影，社会化网络推荐中item是人，这些item属性都不好抽。其实，几乎在所有实际情况中我们抽取的item特征都仅能代表item的一些方面，不可能代表item的所有方面。这样带来的一个问题就是可能从两个item抽取出来的特征完全相同，这种情况下CB就完全无法区分这两个item了。比如如果只能从电影里抽取出演员、导演，那么两部有相同演员和导演的电影对于CB来说就完全不可区分了。
- 无法挖掘出用户的潜在兴趣（Over-specialization）：既然CB的推荐只依赖于用户过去对某些item的喜好，它产生的推荐也都会和用户过去喜欢的item相似。如果一个人以前只看与推荐有关的文章，那CB只会给他推荐更多与推荐相关的文章，它不会知道用户可能还喜欢数码。
- 无法为新用户产生推荐（New User Problem）：新用户没有喜好历史，自然无法获得他的profile，所以也就无法为他产生推荐了。当然，这个问题CF也有。

1.5总结

CB应该算是第一代的个性化应用中最流行的推荐算法了。但由于它本身具有某些很难解决的缺点（如上面介绍的第1点），再加上在大多数情况下其精度都不是最好的，目前大部分的推荐系统都是以其他算法为主（如CF），而辅以CB以解决主算法在某些情况下的不精确性（如解决新item问题）。但CB的作用是不可否认的，只要具体应用中有可用的属性，那么基本都能在系统里看到CB的影子。组合CB和其他推荐算法的方法很多（我很久以后会写一篇博文详细介绍之），最常用的可能是用CB来过滤其他算法的候选集，把一些不太合适的候选（比如不要给小孩推荐偏成人的书籍）去掉。

2、TF-IDF

TF-IDF（term frequency–inverse document frequency，词频-逆向文件频率）是一种用于信息检索（information retrieval）与文本挖掘（text mining）的常用加权技术。

TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

TF-IDF的主要思想是：如果某个单词在一篇文章中出现的频率TF高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

2.1.TF——词频(Term Frequency)

词频（TF）表示词条（关键字）在文本中出现的频率。这个数字通常会被归一化(一般是词频除以文章总词数), 以防止它偏向长的文件。

公式：$$tf_{ij} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

即：$$TF_w = \frac{\text{在某一类中词条}w\text{出现的次数}}{\text{该类中所有的词条数目}}$$

其中 **ni,j** 是该词在文件 **dj** 中出现的次数，分母则是文件 dj 中所有词汇出现的次数总和；

2.2.IDF——逆向文件频率（Inverse Document Frequency）

逆向文件频率 (IDF)：某一特定词语的IDF，可以由总文件数目除以包含该词语的文件的数目，再将得到的商取对数得到。如果包含词条t的文档越少, IDF越大，则说明词条具有很好的类别区分能力。

公式：$$idf_i = \log \frac{|D|}{|\{j : t_i \in d_j\}|}$$

其中，｜D｜**是语料库中的文件总数**。 |{j:ti∈dj}| 表示包含词语 ti 的文件数目（即 ni,j≠0 的文件数目）。如果该词语不在语料库中，就会导致分母为零，因此一般情况下使用 1+|{j:ti∈dj}|

即：$$IDF = log(\frac{\text{语料库的文档总数}}{\text{包含词条}w\text{的文档数} + 1}), \text{分母之所以要加}1\text{，是为了避免分母为}0$$

2.3.TF-IDF实际上是：TF * IDF

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

公式：$$TF - IDF = TF * IDF$$

2.4.应用

（1）搜索引擎；（2）关键词提取；（3）文本相似性；（4）文本摘要

2.5.缺点

TF-IDF 采用文本逆频率 IDF 对 TF 值加权取权值大的作为关键词，但 IDF 的简单结构并不能有效地反映单词的重要程度和特征词的分布情况，使其无法很好地完成对权值调整的功能，所以 TF-IDF 算法的精度并不是很高，尤其是当文本集已经分类的情况下。

在本质上 IDF 是一种试图抑制噪音的加权，并且单纯地认为文本频率小的单词就越重要，文本频率大的单词就越无用。这对于大部分文本信息，并不是完全正确的。IDF 的简单结构并不能使提取的关键词， 十分有效地反映单词的重要程度和特征词的分布情 况，使其无法很好地完成对权值调整的功能。尤其是在同类语料库中，这一方法有很大弊端，往往一些同类文本的关键词被盖。

TF-IDF算法实现简单快速，但是仍有许多不足之处：

（1）没有考虑特征词的位置因素对文本的区分度，词条出现在文档的不同位置时，对区分度的贡献大小是不一样的。

（2）按照传统TF-IDF，往往一些生僻词的IDF(反文档频率)会比较高、因此这些生僻词常会被误认为是文档关键词。

（3）传统TF-IDF中的IDF部分只考虑了特征词与它出现的文本数之间的关系，而忽略了特征项在一个类别中不同的类别间的分布情况。

（4）对于文档中出现次数较少的重要人名、地名信息提取效果不佳。

3、代码详解

3.1导入必要的函数库

```
import pandas as pd
import numpy as np
import csv
```

3.2导入tsv文件

```
#定义一个读取tsv文件的函数
def read_from_tsv(file_path: str, column_names: list) -> list:
    csv.register_dialect('tsv_dialect', delimiter='\t', quoting=csv.QUOTE_ALL)
    with open(file_path, "r",encoding='utf-8') as wf:
        reader = csv.DictReader(wf, fieldnames=column_names, dialect='tsv_dialect')
        datas = []
        for row in reader:
            data = dict(row)
            datas.append(data)
    csv.unregister_dialect('tsv_dialect')
    return datas
```

```
#读取训练集和测试集
train_data = read_from_tsv('train.tsv',['Uid','Date', 'History', 'Impression'])[1:]
train_news_data = read_from_tsv('train_news.tsv',['Nid','Category', 'SubCategory',
'Title', 'Abstract'])[1:]
test_data = read_from_tsv('test.tsv',['Uid','Date', 'History', 'Impression'])[1:]
test_news_data = read_from_tsv('test_news.tsv',['Nid','Category', 'SubCategory',
'Title', 'Abstract'])[1:]
```

```
#合并数据
data = train_data + test_data
news_data = train_news_data + test_news_data
```

```
#随机查看合并训练的数据
train_data[:2]
train_news_data[:2]
test_data[:2]
test_news_data[:2]
```

以下是生成基于内容的DNN模型

3.3生成数据

```
# feature
# 每一个feature实际上是其各种浏览内容的几个部分组成：
#1.最关注的3个Category
#2.Abstract与Title进行词过滤后最热的n个词汇，并进行word emdbedding

# 生成类别集合，方便下面数据转换
news_Category_set = []
for i in train_news_data:
    news_Category_set.append(i['Category'])
for i in test_news_data:
    news_Category_set.append(i['Category'])
news_Category_set = list(set(news_Category_set))
news_Category_set
```

```
#生成feature1, Category
feature_c_m = []
for i in train_data[:200]:
    m = []#中介容器,每一个数据清空一次
    for j in i['History'].split(' '):
        for h in range(len(news_data)):
            if j == news_data[h]['Nid'].strip(' '):
                j = news_data[h]['Category']
            else:
                pass
        m.append(news_Category_set.index(j))
    feature_c_m.append(m)
```

```
# 筛选出出现最高的三个类型
def select_3(arr):
    result = {}
    for i in set(arr):
        result[i] = arr.count(i)
    d_order=sorted(result.items(),key=lambda x:x[1],reverse=False)
    if len(d_order)>=3:
        a =  [d_order[-3][0],d_order[-2][0],d_order[-1][0]]
    else:
        a = [100,100,100]
    return a
```

```
feature_c = []
for i in feature_c_m:
    feature_c.append(select_3(i))
```

```
#生成feature2，词嵌入
feature_w_m = []
for i in data[:200]:
    m = ''#中介容器,每一个数据清空一次
    for j in i['History'].split(' '):
        for h in range(len(news_data)):
            if j == news_data[h]['Nid'].strip(' '):
                a = news_data[h]['Abstract']
                b = news_data[h]['Title']
            else:
                pass
        m = a+b
    feature_w_m.append(m)
```

```
from gensim.models import word2vec
import gensim
import jieba
import jieba.analyse
words_ls = []
for i in feature_w_m:
    words = jieba.analyse.extract_tags(i, topK=5)
    words_ls.append(words)
#model = word2vec.Word2Vec(words_ls,size=12,window=2,min_count=1)
#model.save('moxing.model')
model = gensim.models.Word2Vec.load('moxing.model')
```

3.4搭建模型

```
network = models.Sequential()
network.add(layers.Dense(16, activation='relu'))
network.add(layers.Dense(8))
network.add(layers.Dense(1))
network.build(input_shape=(None,63))
optimizer = tf.keras.optimizers.RMSprop(0.01)
network.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])  # 指定评价指标为准备率
# 模型训练
network.summary()
```

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_27 (Dense)             multiple                  1024
_________________________________________________________________
dense_28 (Dense)             multiple                  136
_________________________________________________________________
dense_29 (Dense)             multiple                  9
=================================================================
Total params: 1,169
Trainable params: 1,169
Non-trainable params: 0
_________________________________________________________________
```

```
# 模型装配
network.fit(db_train, nb_epoch=10)
network.save('my_model.h5')
```

```
Train for 98 steps
Epoch 1/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0223 - mae: 0.0429 - mse: 0.02
23
Epoch 2/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0210 - mae: 0.0423 - mse: 0.02
10
Epoch 3/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0218 - mae: 0.0435 - mse: 0.02
18
Epoch 4/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0209 - mae: 0.0432 - mse: 0.02
09
Epoch 5/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0219 - mae: 0.0427 - mse: 0.02
19
Epoch 6/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0217 - mae: 0.0431 - mse: 0.02
17
Epoch 7/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0215 - mae: 0.0422 - mse: 0.02
15
Epoch 8/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0212 - mae: 0.0426 - mse: 0.02
12
Epoch 9/10
98/98 [==============================] - 0s 3ms/step - loss: 0.0214 - mae: 0.0435 - mse: 0.02
14
Epoch 10/10
98/98 [==============================] - 0s 4ms/step - loss: 0.0212 - mae: 0.0414 - mse: 0.02
12
```

3.5模型应用

```
len(test_data)#2000

n = 5
count=0
ls = []
for i in range(n):
    count+=len(test_data[i]['Impression'].split(' '))
    ls.append(len(test_data[i]['Impression'].split(' ')))

count #142

ls  #[38, 20, 12, 5, 67]
```

```
#生成feature1, Category
feature_c_m = []
for i in test_data[:count]:
    m = []#中介容器,每一个数据清空一次
    for j in i['History'].split(' '):
        for h in range(len(news_data)):
            if j == news_data[h]['Nid'].strip(' '):
                j = news_data[h]['Category']
            else:
                pass
        m.append(news_Category_set.index(j))
    feature_c_m.append(m)

feature_c = []
for i in feature_c_m:
    feature_c.append(select_3(i))

#生成feature2, 词嵌入
feature_w_m = []
for i in test_data[:count]:
    m = ''#中介容器,每一个数据清空一次
    for j in i['History'].split(' '):
        for h in range(len(news_data)):
            if j == news_data[h]['Nid'].strip(' '):
                a = news_data[h]['Abstract']
                b = news_data[h]['Title']
            else:
                pass
        m = a+b
    feature_w_m.append(m)

for i in feature_w_m:
    words = jieba.analyse.extract_tags(i, topK=5)
    words_ls.append(words)
feature_w = []
```

```
for m in words_ls[0:len(train_data[:count])]:
    mid3 = []
    mid3.append(model.wv[m[0]].tolist())
    mid3.append(model.wv[m[1]].tolist())
    mid3.append(model.wv[m[2]].tolist())
    mid3.append(model.wv[m[3]].tolist())
    mid3.append(model.wv[m[4]].tolist())
    feature_w.append(np.array(mid3).reshape(60).tolist())

# 生成label，并产生与之对应的feature
feature_m = []
label_m = []
for i in train_data[:count]:
    #print(train_data.index(i))
    feature_mm = feature_c[train_data.index(i)] + list(feature_w[train_data.index(i)])
    for j in i['Impression'].split(' '):
        feature_m.append(np.array(feature_mm))
        label_m.append(int(j[-1]))

x_test, y_test = get_tensor(feature_m, label_m)
```

```
def p(x):
    y = tf.convert_to_tensor(x)
    z = tf.cast(y,dtype=tf.float32)
    return z

def panduan(x):
    if x>0.5:
        x=1
    else:
        x=0
    return x

result = []
for i in range(142):
    result.append(panduan(network.predict(p([x_test[i]]))))
```

```
n = 0
appends = []
for i in ls:
    appends.append(result[n:n+i])
    n = n + i
appends
```

```
def ls2str(ls):
    m = ''
    for i in ls:
        m+=i
    return m
new_data = []
columns = []
for i in range(len(appends)):
    a = test_data[i]
    a['test'] = ls2str(appends[i])
    new_data[].append(a)
    columns.append(i)
```

3.6写回test.tsv文件

```
def write_to_tsv(output_path: str, file_columns: list, data: list):
    csv.register_dialect('tsv_dialect', delimiter='\t', quoting=csv.QUOTE_ALL)
    with open(output_path, "w", newline="") as wf:
        writer = csv.DictWriter(wf, fieldnames=file_columns, dialect='tsv_dialect')
        writer.writerows(data)
    csv.unregister_dialect('tsv_dialect')
```

```
write_to_tsv('result.tsv',columns,new_data)
```

4、提交文件列表

实验报告lab2-10185102142-李泽浩

源代码CODE.ipynb

写回预测结果后的test.tsv

模型