实体融合

互联网上同一个实体可能来自不同的数据源（站点），它们的名字不完全相同，但指代的确实是同一个实体，虽然实体名字有细微的差别，其实是指代同一个事务，可以利用一定的策略将他们融合归一，通过对不同数据源中的实体信息进行整合，形成更加全面的实体信息。

这里我们将Amazon的众多商品的数据记录文件(Amazon.csv)，与Google对众多商品的数据库记录文件(Google.csv)的数据匹配起来

首先读入所给数据

import pandas as pd

with open("stopwords.txt",'r') as file:

stopwords=file.read()

amazon\_df=pd.read\_csv("Amazon.csv",encoding="ISO-8859-1")

google\_df=pd.read\_csv("Google.csv",encoding="ISO-8859-1")

perfectMap\_df=pd.read\_csv("Amazon\_Google\_perfectMapping.csv",encoding="ISO-8859-1")

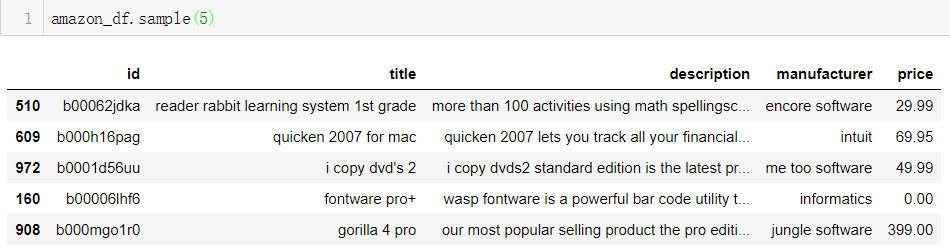
perfectMap=[]

def buildPerfectMap(x): perfectMap.append((x['idAmazon'],x['idGoogleBase']))

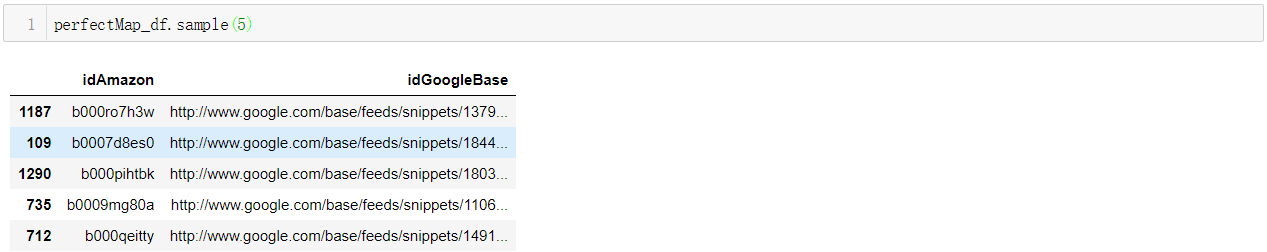
perfectMap\_df.apply(buildPerfectMap,axis=1)

pass

amazon\_df数据如下所示



perfectMap\_df数据如下所示



进行分词操作，并去掉停词，simple\_tokenize()函数转为小写字母并去除标点符号等，tokenize(string)去掉停词

import re

stopwords = stopwords.split('\n')

split\_regex = r'\w+'

def simple\_tokenize(string):

string=string.lower()

return re.findall(split\_regex,string)

def tokenize(string):

if not type(string) is str:return []

xx=simple\_tokenize(string)

for i in stopwords:

while i in xx:

xx.remove(i)

return xx

rec2tok(x,dic)函数用于将输入正规化，键值对，将id作为键。所包含的词为值

def rec2tok(x,dic):

if not type(x['description']) is str:x['description']=''

if not type(x['manufacturer']) is str:x['manufacturer']=''

dic[x['id']]=tokenize(x['title']+' '+x['description']+' '+x['manufacturer'])

调用该方法

amazon\_rec2tok = {}

google\_rec2tok = {}

amazon\_df.apply(lambda x:rec2tok(x,amazon\_rec2tok),axis=1)

google\_df.apply(lambda x:rec2tok(x,google\_rec2tok),axis=1)

可以得到如下结果



统计词频

def inc(i,dic):

if i in dic: dic[i]+=1

else: dic[i]=1

计算TF-IDF值，TF(token)=该token在tokens中的出现次数/tokens中的总token数

def tf(tokens):

tfs={}

for i in tokens: inc(i,tfs)

n=float(len(tokens))

for i in tfs:tfs[i]/=n

return tfs

IDF(token)=tokens(token列表)的个数/出现过该token的tokens(token列表)的个数

def idf(rec2tok):

idfs={}

N=float(len(amazon\_df)+len(google\_df))

for i in rec2tok:

s=set(rec2tok[i])

for j in s:inc(j,idfs)

for i in idfs:idfs[i]=N/idfs[i]

return idfs

TF-IDF:TF-IDF(token)=TF(token)\*IDF(token)

def tfidf(tokens,idfs):

ans=tf(tokens)

s=set(tokens)

for i in ans:

ans[i]\*=idfs[i]

return ans

返回键值对，键为词，值为含有该词的id列表，参数为由rec2tok(x,dic)函数得到的id与对应词列表的字典

def invertIndex(forward\_index):

#return a mapping from token to list-of-record-IDs

ans={}

for i in forward\_index:

for j in forward\_index[i]:

if j in ans:ans[j].append(i)

else: ans[j]=[i]

return ans

调用该方法，

amazon\_inv=invertIndex(amazon\_rec2tok)

可以得到如下结果



累加求和

def dotprod(a, b):

ans=0

for i in a:

if i in b: ans+=a[i]\*b[i]

return ans

向量的模

def norm(a):

ans=0

for i in a:

ans+=a[i]\*\*2

return math.sqrt(ans)

计算每个词总的idf值

from collections import Counter

idfs\_full = dict(Counter(idf(amazon\_rec2tok))+Counter(idf(google\_rec2tok)))

计算每个id对应的词的权重

google\_weights={i:tfidf(google\_rec2tok[i],idfs\_full) for i in google\_rec2tok}

amazon\_weights={i:tfidf(amazon\_rec2tok[i],idfs\_full) for i in amazon\_rec2tok}

计算每个id对应的模的大小

google\_norm={i:norm(google\_weights[i]) for i in google\_weights}

amazon\_norm={i:norm(amazon\_weights[i]) for i in amazon\_weights}

定义buildSim() 用于计算每两个对应之间的相似度

def buildSim(Id,weight,norm,weights,norms,inv,sims):

#weights : Id->token->weight

#norms : Id->norm

for i in weight:

if i in inv:

for j in inv[i]:

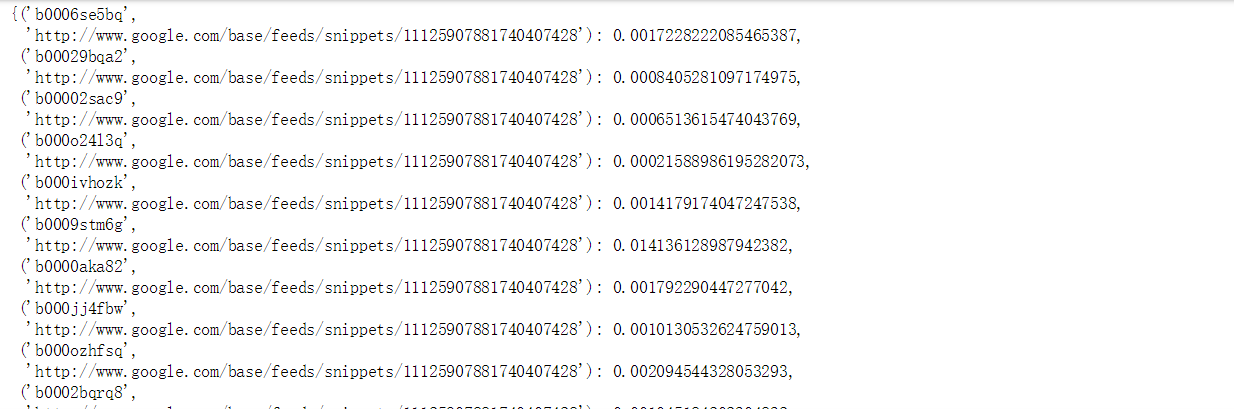
if not (j,Id) in sims: sims[(j,Id)]=dotprod(weight,weights[j])/norm/norms[j]

sims={}

for i in google\_weights:

buildSim(i,google\_weights[i],google\_norm[i],amazon\_weights,amazon\_norm,amazon\_inv,sims)

得到的sims如下



计算准确率并作图

true\_dup\_sims = []

def truepos(threshold):

global true\_dup\_sims

true\_dup\_sims=[]

for i in sims:

if sims[i]>threshold:

true\_dup\_sims.append(i)

def bin(similarity):

return int(similarity \* nthresholds)

def falsepos(threshold):

ans=0

for i in true\_dup\_sims:

if not i in perfectMap: ans+=1

return ans

def precision(threshold):

truepos(threshold)

a=len(true\_dup\_sims)-falsepos(threshold)

b=len(true\_dup\_sims)

return a\*1.0/b

nthresholds=100

thresholds = [float(n) / nthresholds for n in range(2, nthresholds)]

p=[precision(n) for n in thresholds]

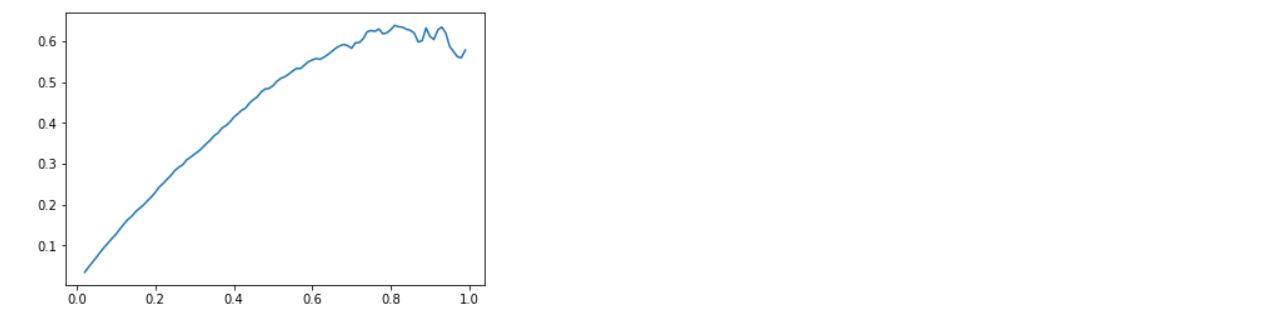
%pylab inline

plt.plot(thresholds,p)

for i in range(0,98):

if p[i]==max(p): print ("最大准确率阈值",thresholds[i])

print ("最大准确率：",max(p))



进一步拓展，想在此基础上探究一下原著小说与相对应的电影时间的相似度，数据从自豆瓣电影与豆瓣图书获取。

首先，寻找数据，发现如下书单：被曾改编成电影的书



编写代码爬取数据

# coding: utf-8

import re

import pandas as pd

import requests

from urllib import parse

from urllib import request

import time

header={

'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36',

'Connection':'keep-alive'

}

def get\_url():

url\_li = []

for i in range(9):

url ='https://www.douban.com/doulist/16224/?start='+str(i\*25)+'&sort=time&playable=0&sub\_type='

html = requests.get(url=url,headers=header).content.decode('utf-8')

movie\_item = re.findall(r'<div class="source">(.\*?)<img', html,re.S)

for i in movie\_item:

url\_li=url\_li+(re.findall(r'<a href="(.\*?)"', i,re.S))

time.sleep(1)

return url\_li

def get\_details(url):

html = requests.get(url=url,headers=header).content.decode('utf-8')

name = re.findall(r'property="v:itemreviewed">(.\*?)<', html,re.S)[0]

description\_raw = re.findall(r'<div class="intro">(.\*?)<\/div>', html,re.S)[0]

li = description\_raw.split("<p>")

description=[]

for i in li:

if(len(i.split('</p>'))>1):

description.append(i.split('</p>')[0])

description = ''.join(description)

return name,description

def find\_book():

comments = pd.DataFrame(columns=['url','name','description'])

comments.to\_csv('book.csv',index=False,encoding="utf-8")

url\_li = get\_url()

for i in url\_li:

try:

name,description = get\_details(i)

print(name)

comments = comments.append({'url':i,'name':name,'description':description},ignore\_index=True)

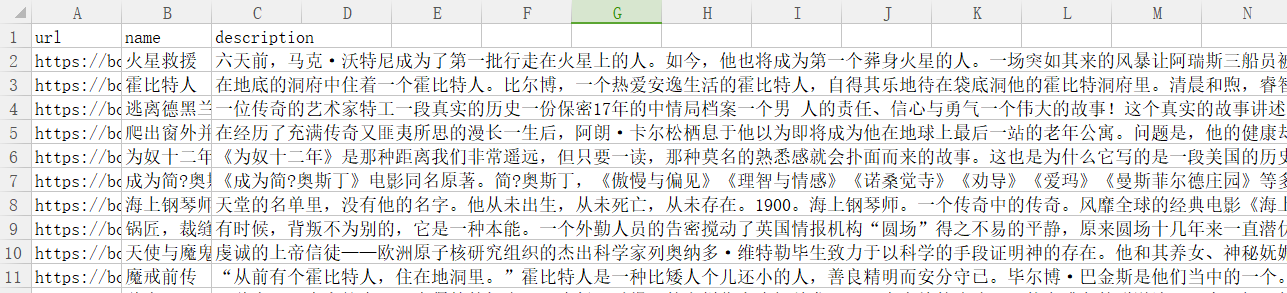
except:

print(i)

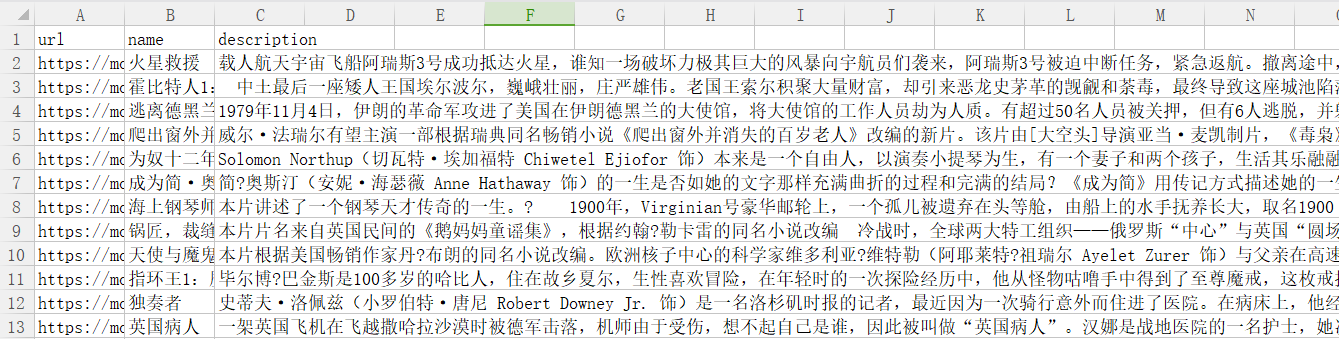
comments.to\_csv('book.csv',index=False,mode='a', header=False,encoding="utf-8")

find\_book()

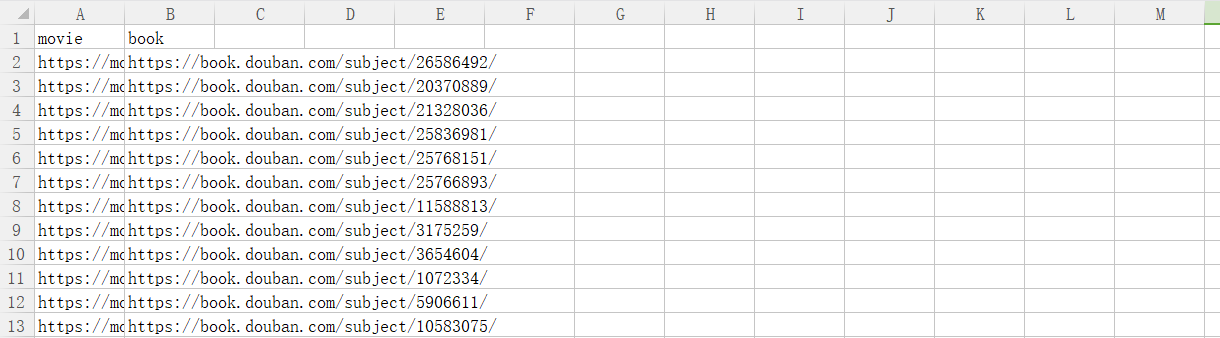
将爬取好的数据存入csv表格中以备使用，内容大致如下所示，包含url、书名、内容简要



手动找到相应的电影，存入



对应关系



分词，

import jieba

def tokenize(string):

stopwords = stopwordslist('stop.txt')

seg\_list = jieba.cut(string)

re\_li = []

for i in seg\_list:

if (not i in stopwords) and len(i)>1:

re\_li.append(i)

return re\_li

def rec2tok(x,dic):

#x:a record from on DataFrame

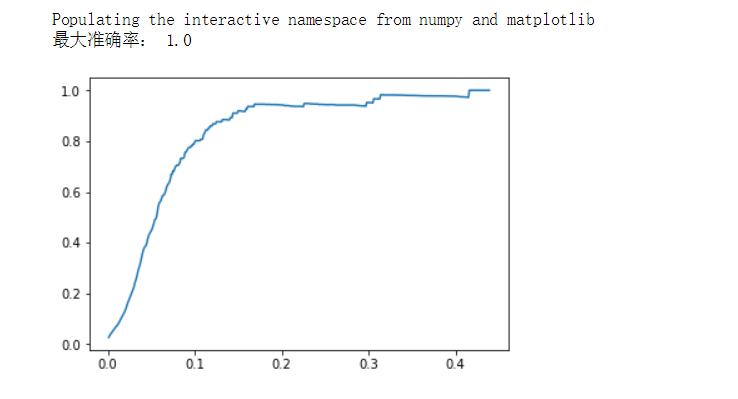
#dic:dictionary that build mappings from record id to tokens

if not type(x['description']) is str:x['description']=''

if not type(x['name']) is str:x['name']=''

dic[x['url']]=tokenize(x['name']+' '+x['description'])

与示例代码类似 最终可得到如下结果（具体代码见附件）



我们可以发现，当阈值设置为0.4时，准确率可以几乎达到100%，我们可以分析的到，因为人名以及一些具体的意象（如火星，特工的等词汇）的原因导致了这样的结果