****

大数据综合应用实践

报告

|  |  |
| --- | --- |
| 题 目： | 作者重名识别 |
| 学生姓名： | 刘宣辰 |
| 指导教师： | 高建良 |
| 学 院： | 计算机学院 |
| 专业班级： | 大数据1702 |

2020年7月

# 1.数据介绍

## 1.1输入数据

下载链接：<https://static.aminer.cn/misc/na-data-kdd18.zip>

原始数据集为json格式，数据集由多篇文章的json数据组成，数据的格式如下：

{  
 **"5b5433eae1cd8e4e150354ca"**: {  
 **"authors"**: [  
 {  
 **"name"**: **"Runzhi Zhang"**,  
 **"org"**: **"Institute of Zoology"**,  
 **"id"**: **"5b5433e6e1cd8e4e15f992c7"** },  
 {  
 **"name"**: **"Hongbin Liang"**,  
 **"org"**: **"Institute of Zoology"**,  
 **"id"**: **"5b5433f3e1cd8e4e1516badf"** },  
 {  
 **"name"**: **"Li Ren"**,  
 **"org"**: **"Institute of Zoology"**,  
 **"id"**: **"5b5433e7e1cd8e4e15fae6ab"** },  
 {  
 **"name"**: **"Guangxue Zhang"**,  
 **"org"**: **"Institute of Zoology"**,  
 **"id"**: **"5b5433f8e1cd8e4e151fdcc5"** }  
 ],  
 **"title"**: **"Induced life cycle transition from holocycly to anholocycly of the Russian wheat aphid (Homoptera: Aphididae)"**,  
 **"keywords"**: [  
 **"Induced transition"**,  
 **"Life cycle pattern"**,  
 **"Russian wheat aphid (diuraphis noxia mordvilko)"** ],  
 **"venue"**: **"Science in China, Series C: Life Sciences"**,  
 **"year"**: 2001,  
 **"abstract"**: **"The Russian wheat aphid (RWA), Diuraphis noxia (Mordvilko), exists with holocyclic life cycle in Tacheng, Xinjiang in Northwest China. It produces males and oviparae to mate and oviposit for overwintering by eggs. Under laboratory conditions with 14 h/d photophase and temperature not lower than 15°C, RWA occurred in parthenogenesis and produced no males. The laboratory populations of Russian wheat aphid, which were kept under natural conditions in fall by 15th, 49th and 81st generation while wild populations produced males and oviparae for mating, produced males and oviparae with their number decreased gradually, but viviparae and nymphs increased sequentially. As a result, it produced a small amount of oviparae and no males emerged in fields by 49 generations' reproduction in laboratory. After development of 81 generations, oviparae happened occasionally and no eggs occurred for overwintering instead of viviparae and nymphs. A hypothesis of RWA disastrous process was proposed. The life cycle of RWA can be changed from holocycly to anholocycly in its long-term spread and evolution. Anholocycly is more dangerous than holocycly to small grains for its strong adaptability and dispersal ability."** }

}

最外层的key值为paperid，最外层的value值为论文信息，包含**authors**（作者信息）、**title**（论文标题）、**keywords**（关键词）、**venue**（期刊/会议名称）、**year**（发表时间）、**abstract**（论文摘要信息）。其中**authors**为一个list列表，包含若干个以json格式存储的作者的信息，包括**name**（作者名）、**org**（所属机构）、**id**（作者真实id/标签值）。

## 1.2输出数据

附件中的output\_example.json是指导书提供的示例输出文件。学生需按照输出数据格式对模型的识别结果进行整理，并基于输出结果和输入数据中的真实标签值评测模型在测试集上的性能，将评测结果输出到csv表格或其他格式的文档中。

输出数据为json格式，其中key值为作者名，value值为重名识别结果，形式为不定长的多维数组，示例如下：

{

**"hongbin\_liang"**:

[

[**"5b5433f4e1cd8e4e151980a3",**

**"5b5433f1e1cd8e4e15122f8b",**

**"5b5433f7e1cd8e4e151f58da",**

**"5b5433ebe1cd8e4e1503d368",**

**"5b5433e5e1cd8e4e15f8537e",**

**"5b5433f7e1cd8e4e151e9907",**

**"5b5433eae1cd8e4e1501c8b3",**

**"5b5433f3e1cd8e4e1517230f",**

**"5b5433e9e1cd8e4e15012ea8"**

],

[**"5b5433e6e1cd8e4e15f9c81e",**

**"5b5433f7e1cd8e4e151e8746",**

**"5b5433e8e1cd8e4e15fdce2e"**

],

[**"5b5433e6e1cd8e4e15f91559",**

**"5b5433ede1cd8e4e1509c56c",**

**"5b5433f5e1cd8e4e151a1a9e",**

**"5b5433e4e1cd8e4e15f5a382",**

**"5b5433eae1cd8e4e150399a1",**

**"5b5433e7e1cd8e4e15fca048"**

],

["**5b5433f5e1cd8e4e151b8329",**

**"5b5433f5e1cd8e4e151ae703",**],

[**"5b5433e6e1cd8e4e15facacb"**]

]

}

对存在重名问题的作者名为**"hongbin\_liang"**的识别结果中，key值为**"hongbin\_liang"**，value为不定长的多维数组。其中每个数组中元素由作者集合中包含作者名**"hongbin\_liang"**的paperid组成，预测同一个数组中的paperid对应的论文属于现实世界中名为**"hongbin\_liang"**的一个人。**特别说明**：多维数组的不同数组之间交集为空。value的多维数组维度表示模型预测真实世界中名字为**"hongbin\_liang"**的作者的人数。

# 2.方法介绍

## 模型概述

该模型的实现基于Spark，以一个作者名*a*为例，设*a*对应的论文集合为，首先通过GraphFrames构建一个全连接的论文网络，网络中的每个节点都代表一篇论文，通过GraphFrames提供的API对网络中的任意2个节点与计算不同特征的相似度，并通过一个二分类器判断与是否属于同一个作者，属于则将的标签设置为1，否则置为0；最后将网络中所有标签为1的边构成的连通块输出即为作者名*a*对应的论文划分结果。模型的具体步骤和组成部分如下。

## 2.1特征工程

特征工程的目的是将论文的文本特征转换成易于计算的形式，比如通过Word2Vec或其他词嵌入模型将数据集中的文本特征转换为向量表示。首先需要对数据集中的文本数据进行清洗，删除或替换文本中的特殊字符，例如将作者姓名中可能存在的“-”和S“.”替换成“\_”，将文本统一转换成小写等。

数据集经过特征工程转换的结果可以有两种形式。第一种是每篇论文都单独对应一个特征向量，在计算相似度时，直接对不同论文的特征向量计算余弦相似度或欧氏距离。第二种是每篇论文的不同属性对应不同特征，例如论文的标题与论文中作者的机构可以分别对应2个词嵌入向量，论文的合作者名字对应一个字符串集合，在计算相似度时，对不同类型的特征采用不同的计算方式，例如向量可以采用余弦相似度计算，字符串集合可以采用jaccard系数计算。本指导书实现的是第二种转换方式，每篇论文都有text，org，coauthor，year共4种特征，其中text是标题与摘要的词嵌入向量，org是作者机构的词嵌入向量，coauthor是合作者名字集合，year是论文发表年份。

## 2.2构建网络

以一个作者名*a*为例，设*a*对应的论文集合为，首先通过GraphFrames构建一个全连接的论文网络，即网络中任意2个节点都存在一条边连接。网络中的每个节点都代表一篇论文。节点的特征对应特征工程得到的论文特征。

## 2.3分类判别

通过GraphFrames提供的API对网络中的任意2个节点与计算不同特征的相似度，将，，，， ，， ，7个相似分数组成的向量作为一个二分类器的输入，二分类器输出0或1，1代表与属于同一个作者，0代表与不属于同一个作者，将的标签设置为二分类器的输出结果，以边的标签为1为过滤条件从中生成子图，并通过GraphFrames提供的connectedComponents()方法输出子图中的连通块，每一个连通块包含的论文都视为同一个作者所写，即得到了作者名*a*对应论文集合的划分结果。

跟老师模型的唯一区别是我将title，venue，abstract和keywords从text一个特征中分出来了，比如逻辑回归，如果只有四个特征，那其参数只有四个，训练数据一旦较大就很难划分特征空间，所以这里将合并的text特征分开，让分类器自动调整参数确定每个特征的权重。

## 2.4评价指标

使用基于pairwise的precision（准确率）、recall（召回率）以及F-measure（F1值）来评测重名作者的识别效果。正确率和召回率的取值范围是[0,1]，这两个分数的值越接近1，表示正确率或召回率越高。F1值则是正确率和召回率的简单调和平均数。pairwise在本次实验中指辨识结果中同一个数组中的论文对pair，pair的定义如下：

设作者名为*a*的作者对应的论文集合为，中的论文根据模型输出中的多维数组生成作者名*a*有关的论文对pair集合：

本文的F1值根据以下定义和公式计算得出：

1. ，正确识别的作者对数量
2. ，错误识别的作者对数量
3. ，未识别出的作者对数量

举例说明：科技文献数据库中的6篇文献A、B、C、D、E、F的**authors**中包含有相同名字的作者名称，其中A、B、C论文中的重名作者的真实标签（**id**）相同，均为label1，D、E的真实标签（**id**）相同，均为label2，F的真实标签（**id**）为label3。则正确的论文对有 (A,B) (A,C) (B,C) (D,E)

假设在预测模型的结果中，A、B、C、D的预测标签相同，E、F预测标签相同。模型预测该作者在现实世界中为2个人。模型输出的预测结果中论文对有(A,B) (A,C) (A,D) (B,C) (B,D) (C,D) (E,F)

模型正确识别的论文对有 (A,B) (A,C) (B,C)，则TP=3

模型错误识别的论文对有 (A,D) (B,D) (C,D) (E,F)，则FP=4

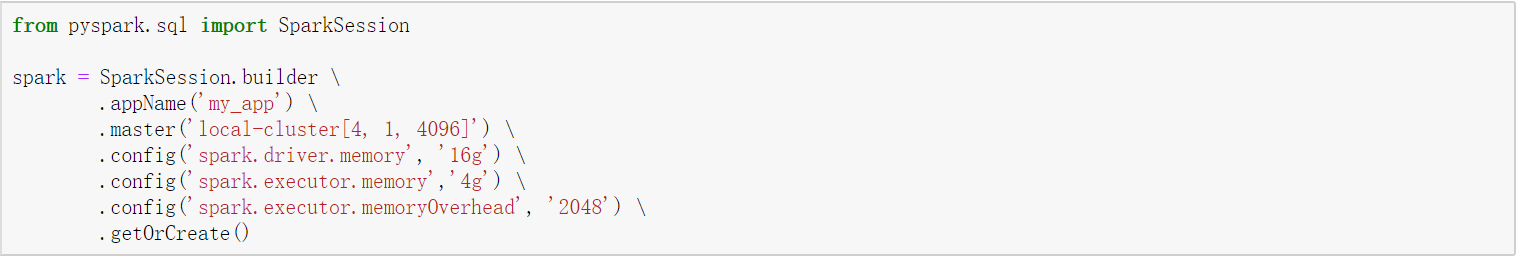
模型未识别出的论文对有 (D,E) ，则FN=1

则precision=3/(3+4)=3/7 recall=3/(3+1)=3/4

实验中自行划分训练集和测试集，假使测试集是100个作者名对应的重名作者数据集，macro-P、macro-R表示100个作者名字对应的precision、recall值的平均值。Macro-F1值表示macro-P和macro-R的调和平均值。

# 3.实验步骤与结果分析

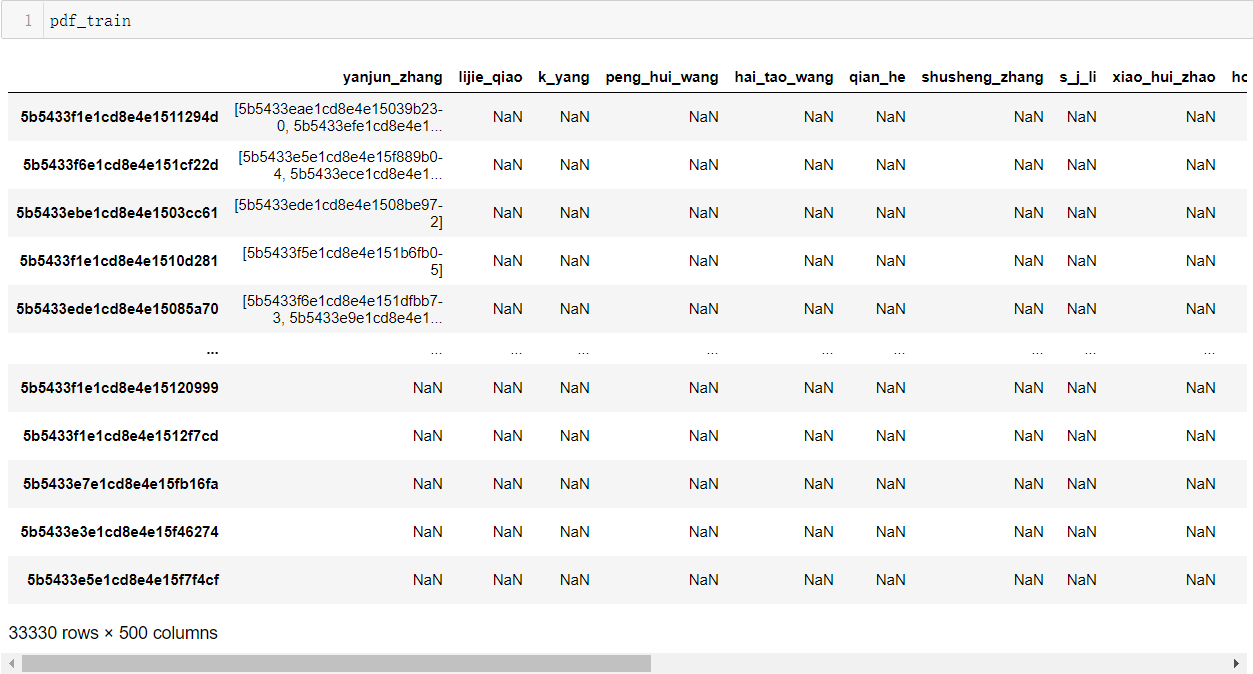
启动spark，通过sparksession将spark 的参数传入。这里master是设置spark 的部署方式，单机的部署方式有3种，Local[\*]本地模式，local-cluster本地伪分布式集群， standalone单机伪分布式集群。这里采用local-cluster模式，其有三个参数local-cluster[numSlaves, coresPerSlave, memoryPerSlave]，numSlaves：模拟集群的Slave节点个数，coresPerSlave：模拟集群的各个Slave节点上的内核数，memoryPerSlave：模拟集群的各个Slave节点上的内存大小。

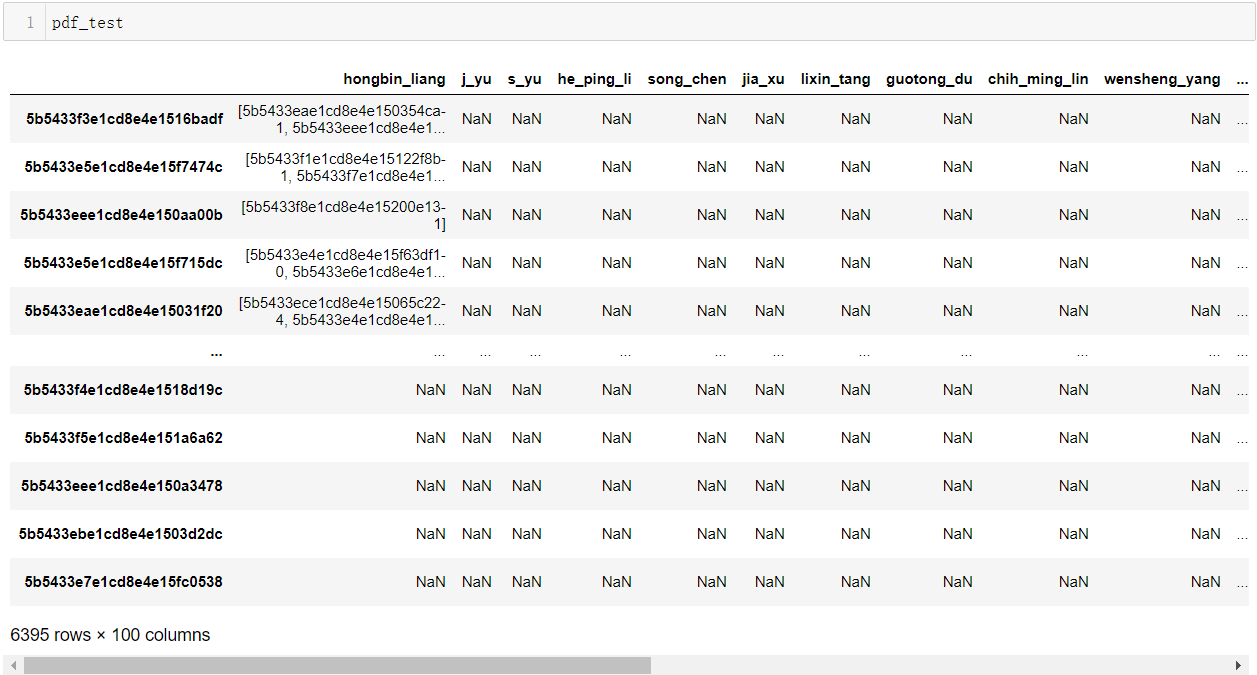


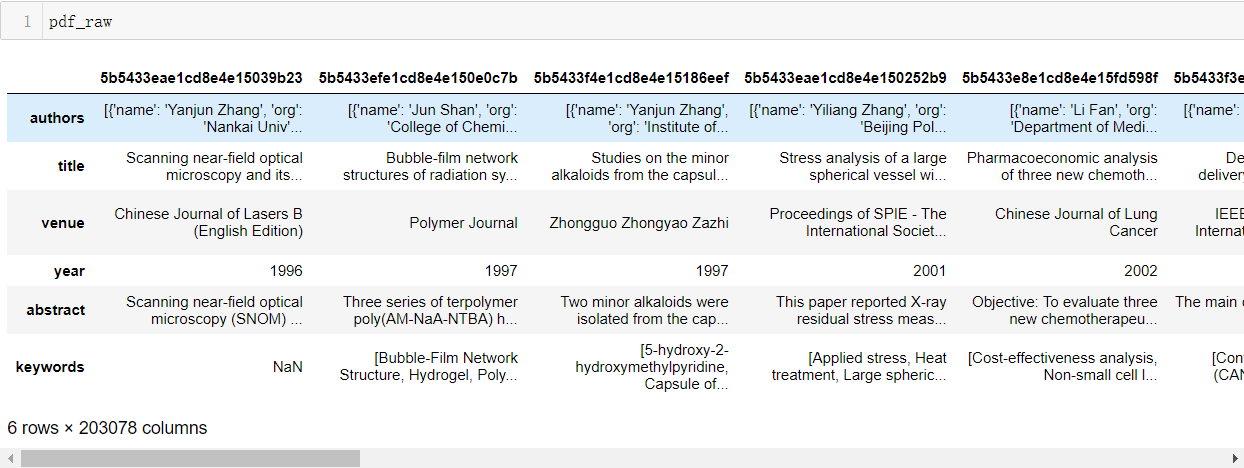
由于数据是json格式，我分别尝试了spark的dataframe和pandas的dataframe中导入json文件的方法，但是由于title中有一些特殊符号，需要特别传入使用utf-8编码读取，所以使用对json支持更好的json库。



查看导入的dataframe：pdf\_train， pdf\_test， pdf\_raw



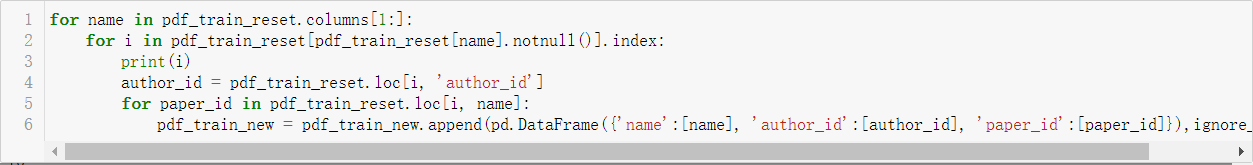




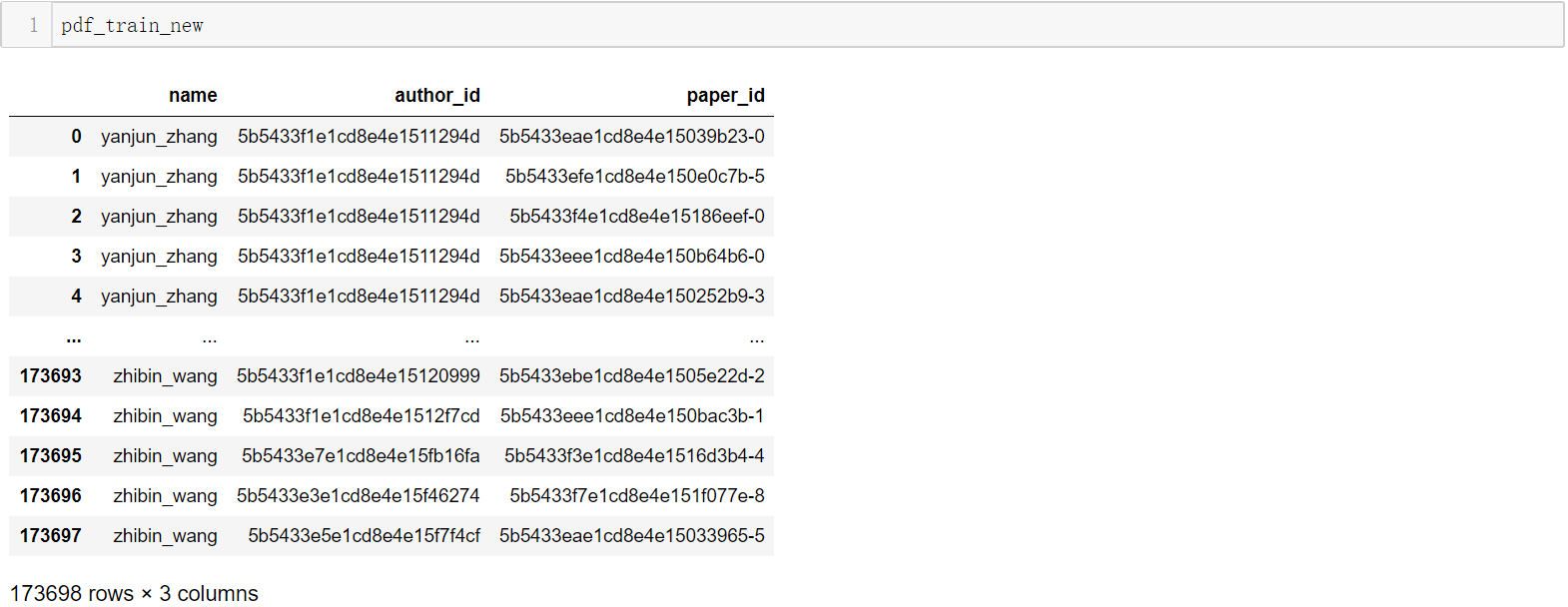
由于训练集和测试集的数据格式相同，对其做的数据处理工作相同，所以后面只展开描述对训练集所做的工作。

观察pdf\_train数据后发现数据格式非常不规整，所以我将其先转置，然后以论文id为主键，将pdf\_train进行了调整。





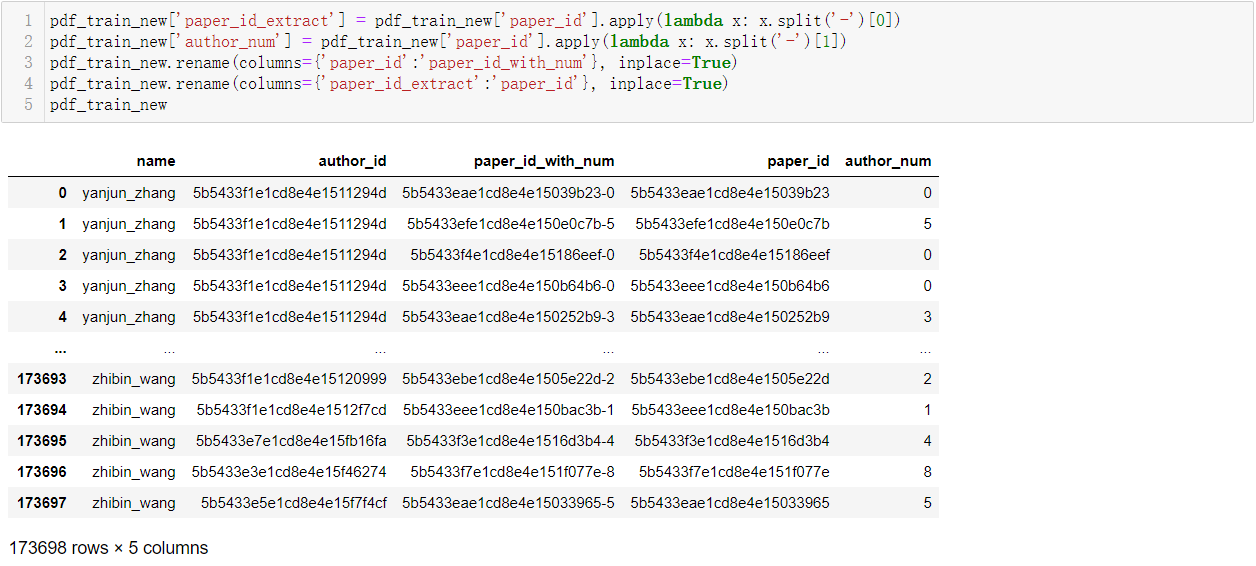
调整后结果如下



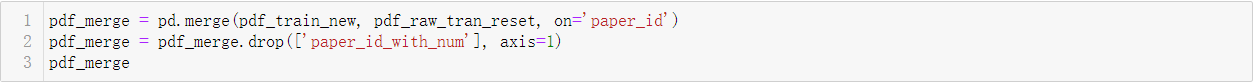
对pdf\_raw进行调整，调整后其结果如下。

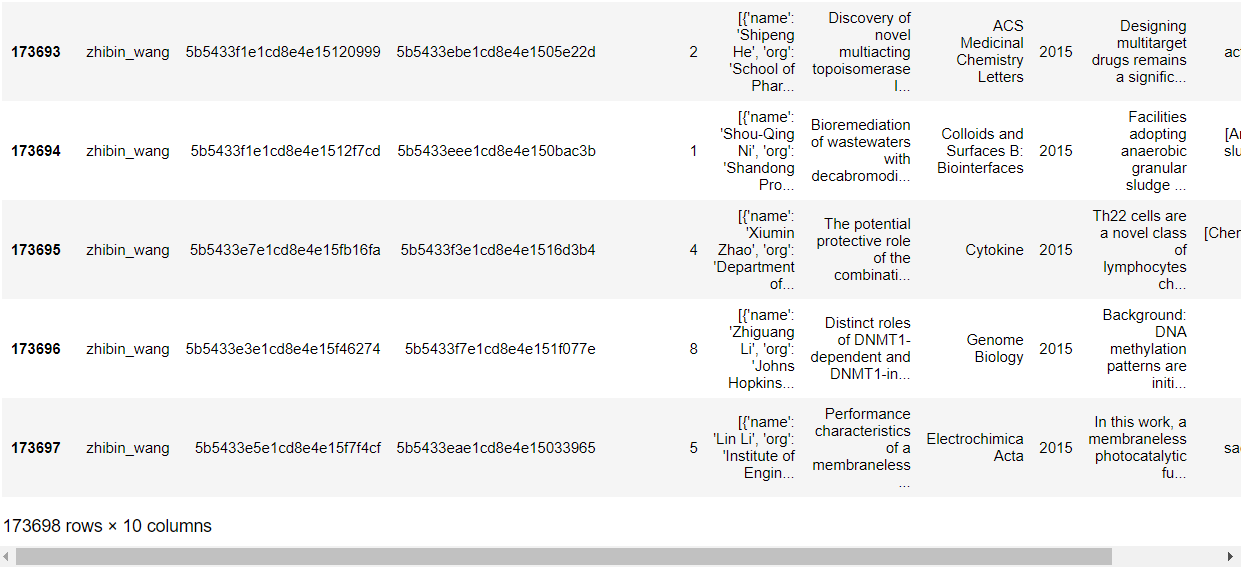
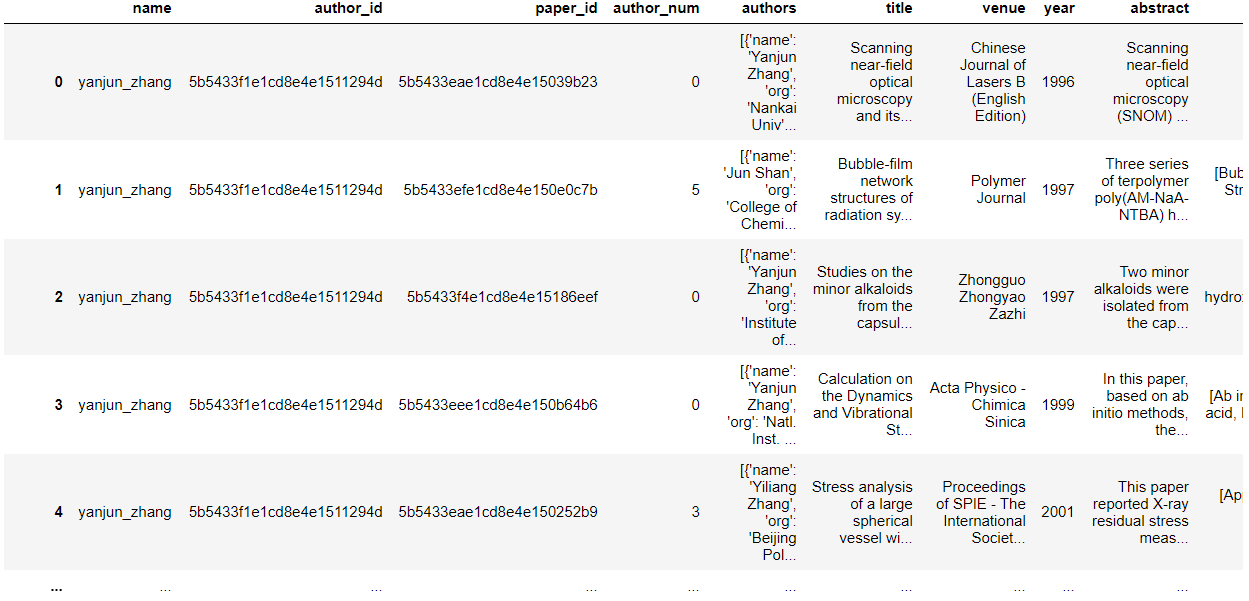


观察发现pdf\_train的paper\_id列与pdf\_raw中的paper\_id列不同，pdf\_train中的paper\_id多了第几作者的后缀，所以为了后面将两个dataframe merge工作，对其调整。

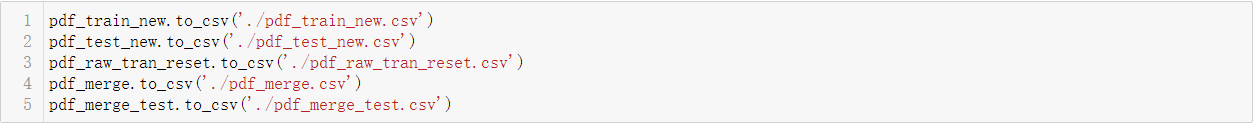


然后进行两个dataframe的merge





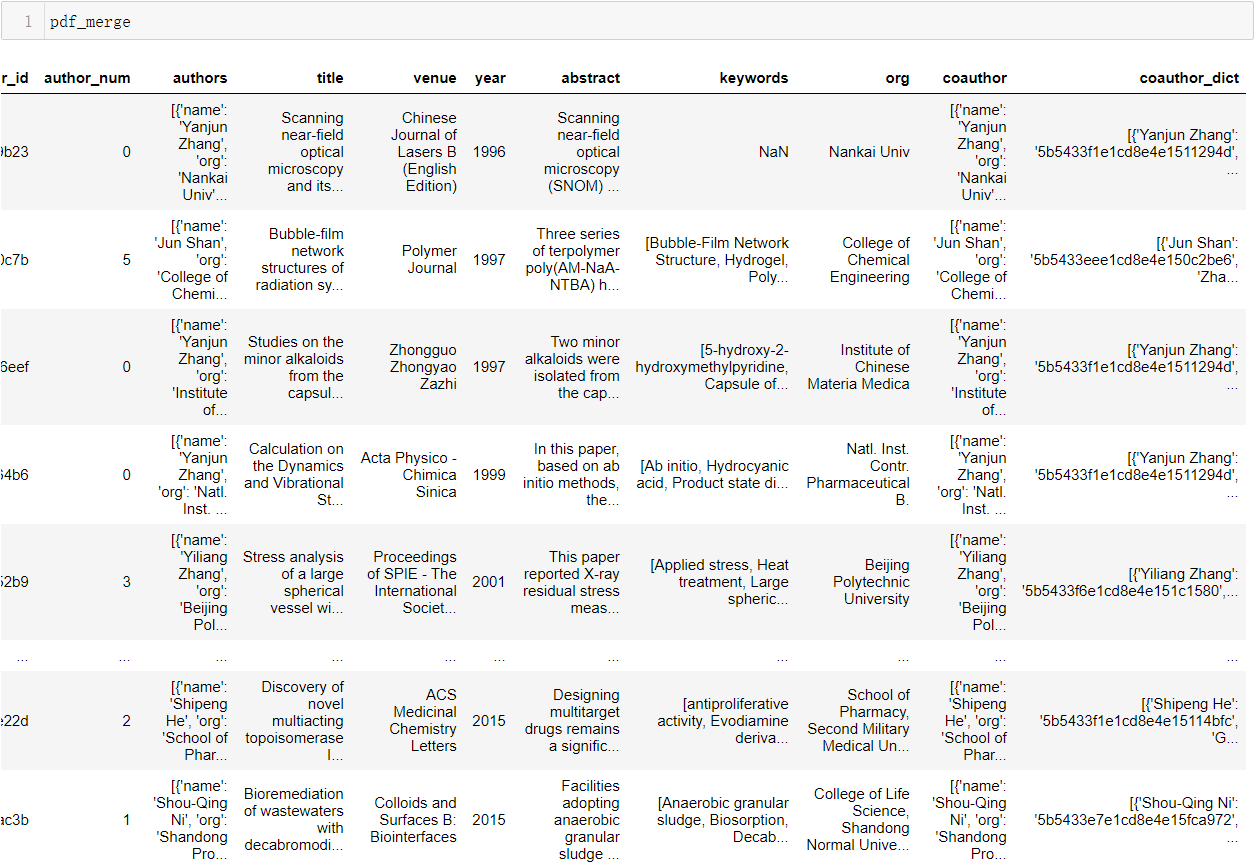
至此，数据初步处理完毕，将结果保存为csv文件。



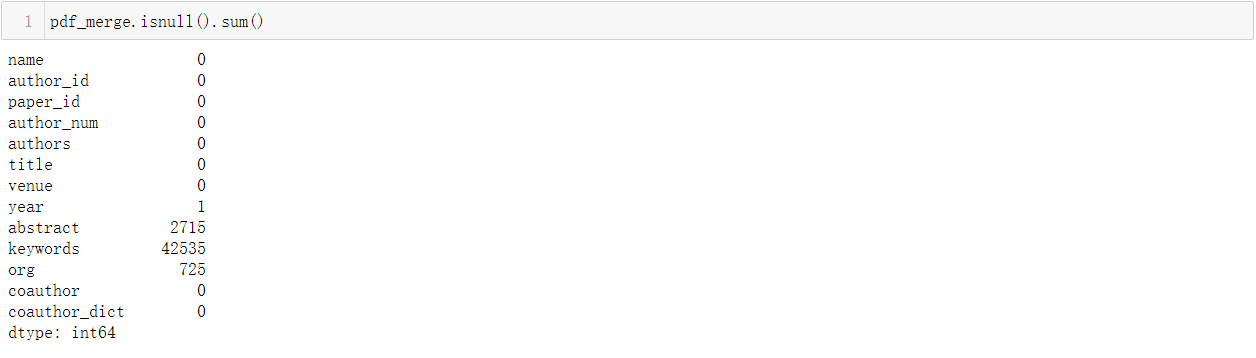
然后我们需要从authors列提取出org和coauthor，由于coauthor也会有重名的情况，所以多加一列coauthor\_dict保留其id用于区分重名作者。



处理后发现多加了三列。



观察发现有keywords有不少缺失值。

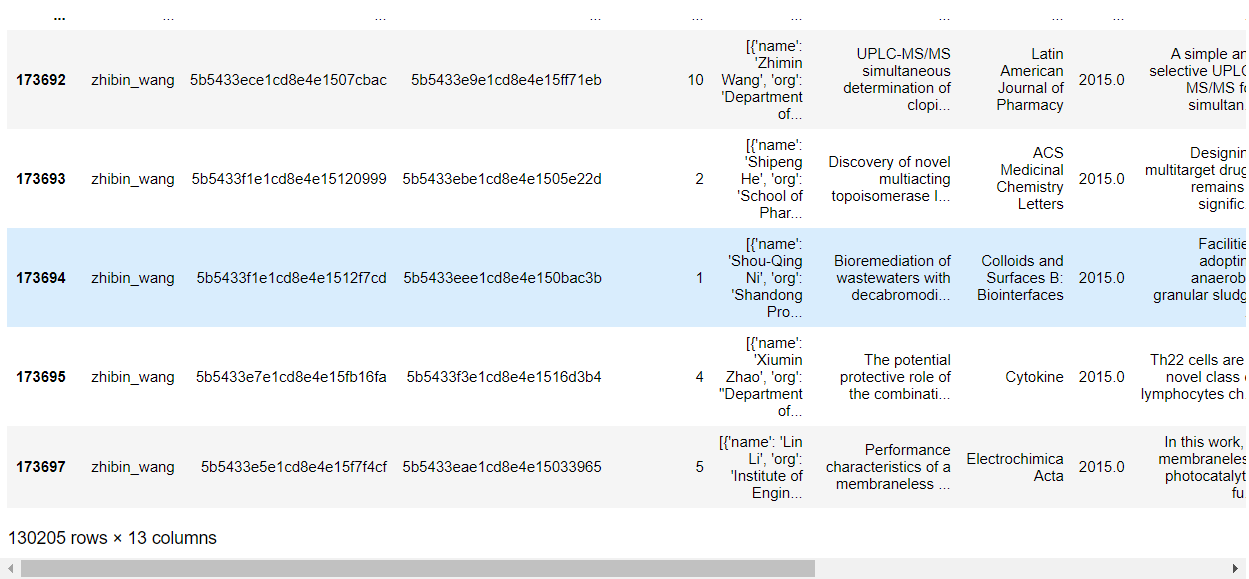


保存pdf\_merge



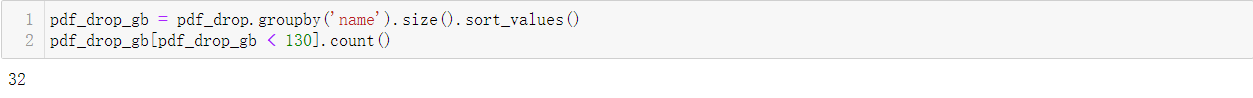
对于缺失值，由于训练集数据过于庞大，而我采用的本地伪分布式部署方式，性能较差，之后训练时不会取到所有数据，所以这里简单去掉有缺失值

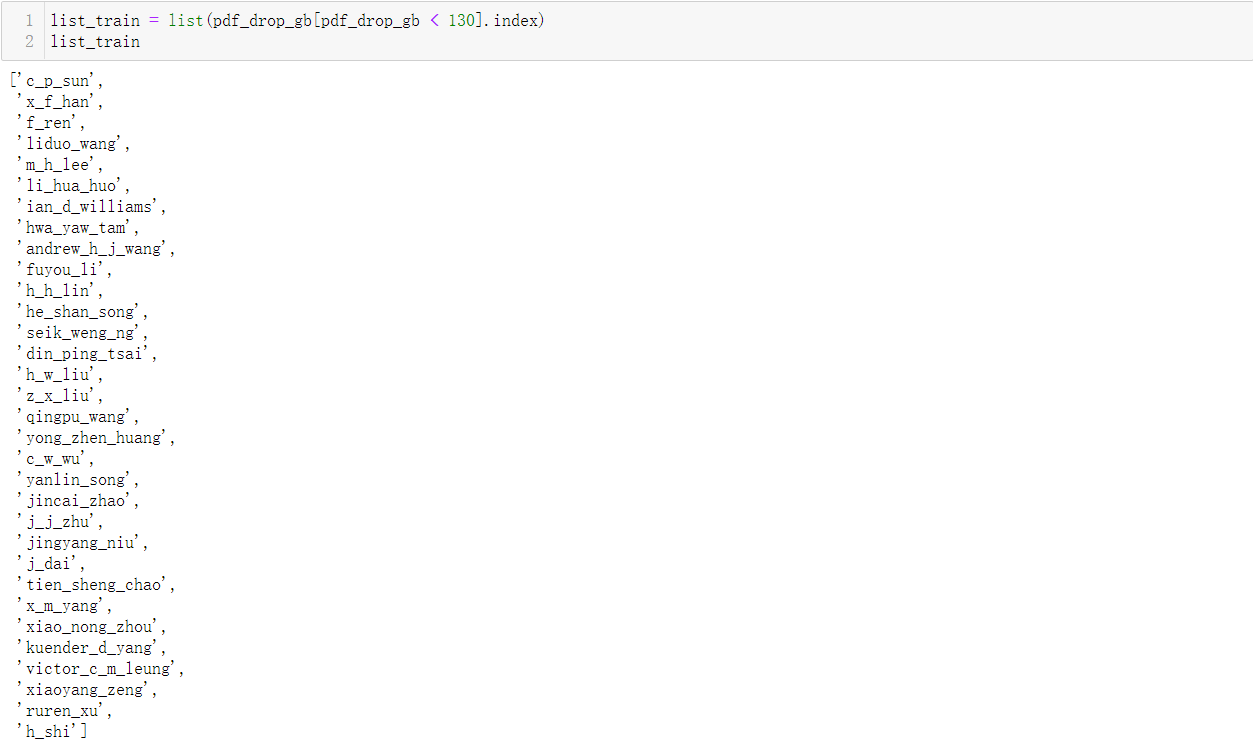




保留了13万左右的记录

由于后面有对构建全连接图Gpa的操作，而这个操作是O()的复杂度，所以尽可能选择重名论文数量少的名字。这里进行排序后，选择所有重名论文小于130的的名字。

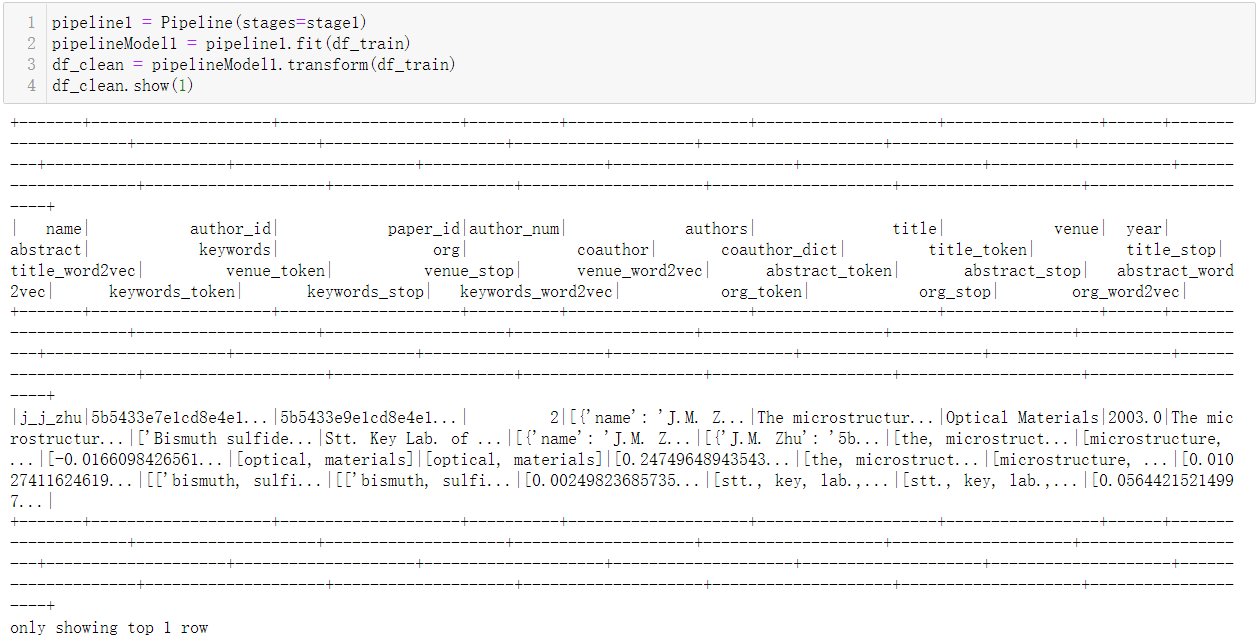


至此，训练集初步处理完毕。

之后进行分词，去停用词，得到词嵌入向量三个步骤。得到经过pipeline处理后dataframe。

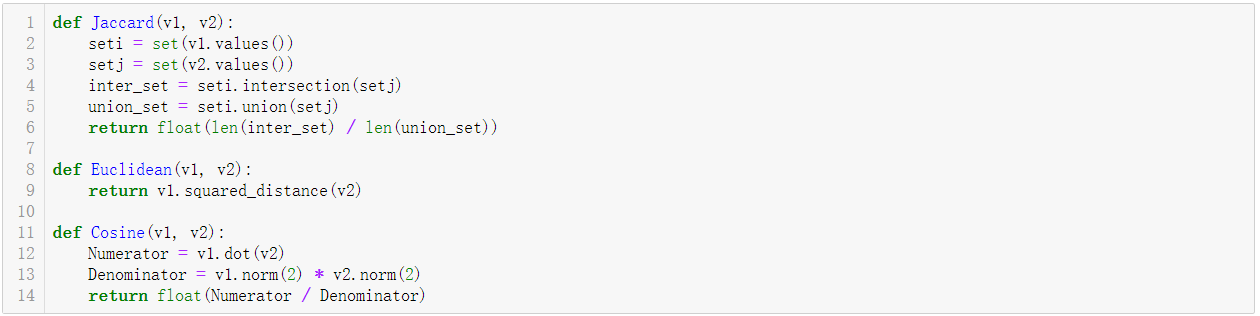






有了词嵌入向量，之后进行相似度计算。

定义相似度计算函数

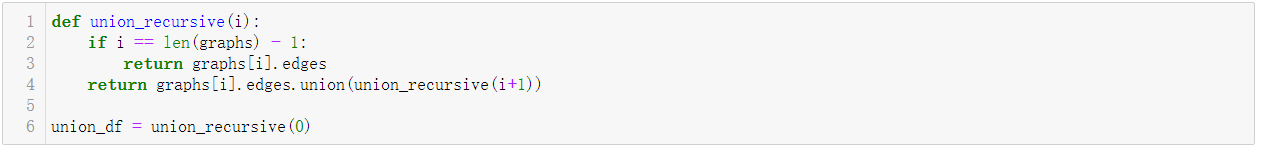


建立全连接图并计算相似度，将相似度增加到边属性上。

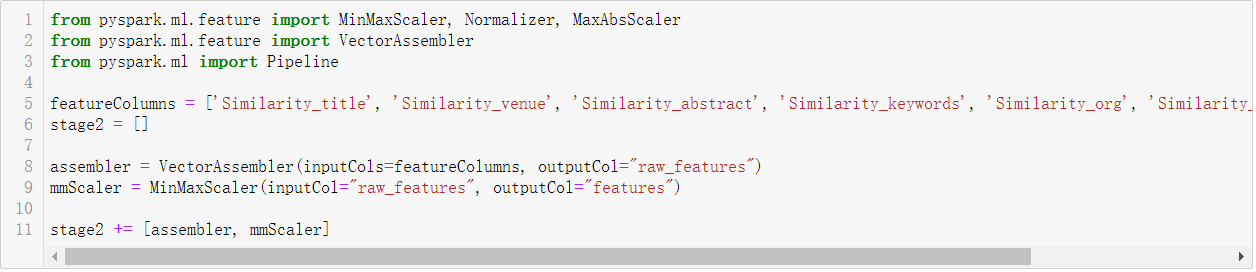




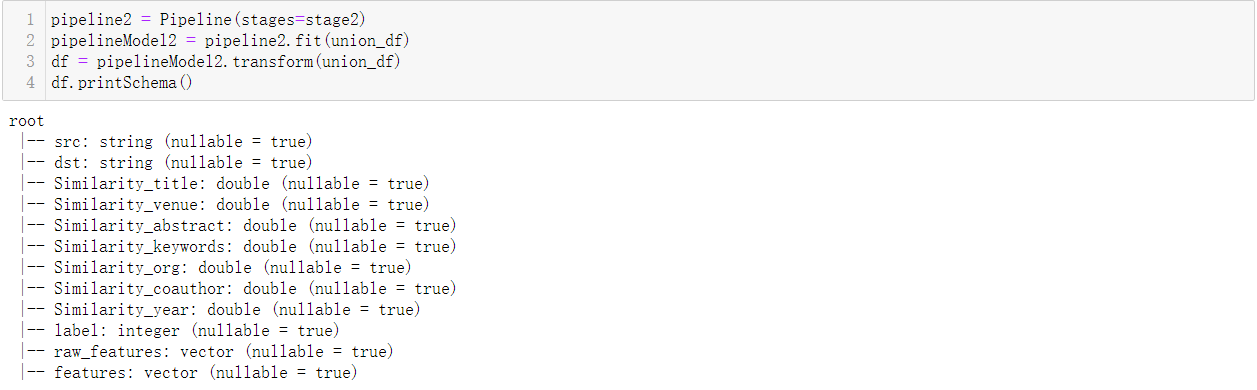
由于每个名字分别建立一个图，但是我们训练时需要一个完整的dataframe，所以需要将所有图联合成一个，由于union方法只能联合两个图，但是我们有32个，所以用递归的方法，从0开始递归联合到边界终止



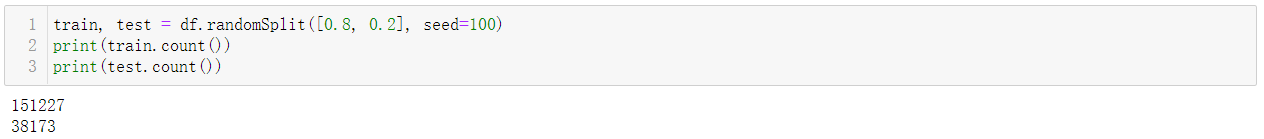
得到完整的训练集后，需要将所有特征assemble到一个特征向量才能继续后面的训练。这里加上了最大最小值归一化，因为每个特征的量纲不同，归一化消除了量纲大小对特征重要性的影响。



虽然有缺失值，但并不影响训练过程。



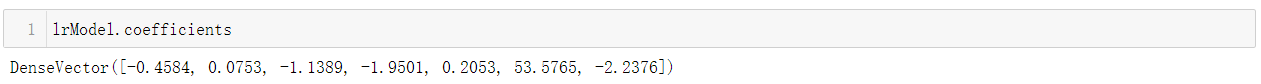
划分训练集和测试集



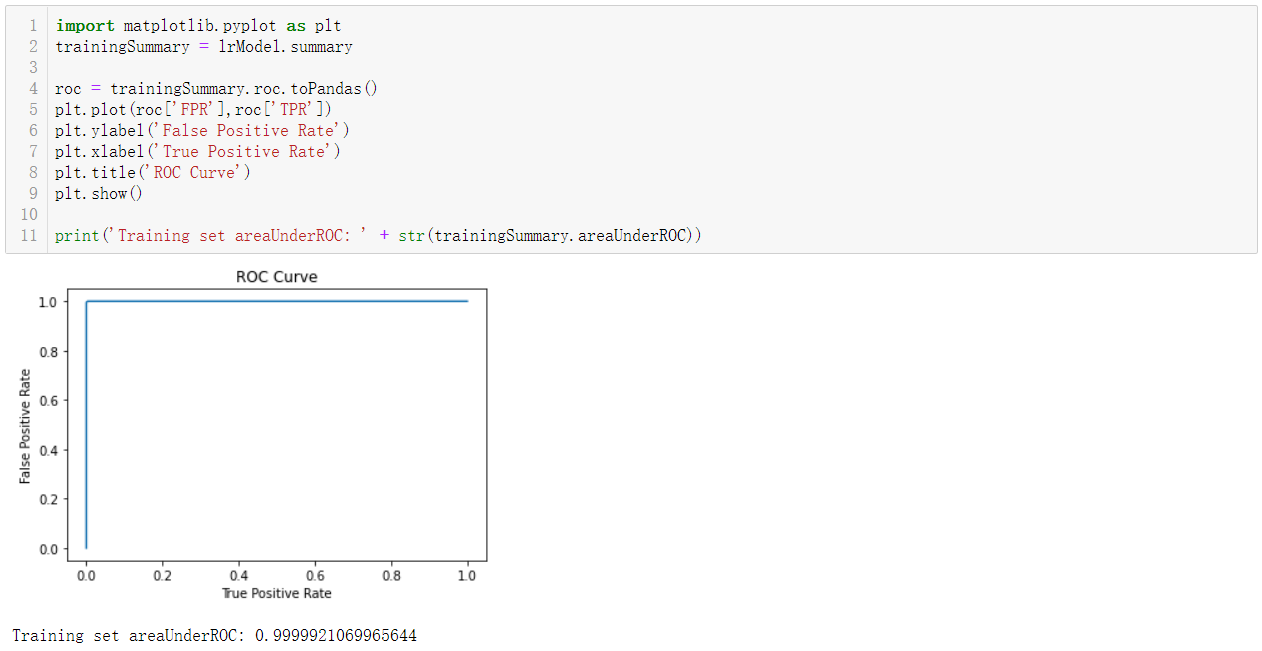
建立分类器并训练

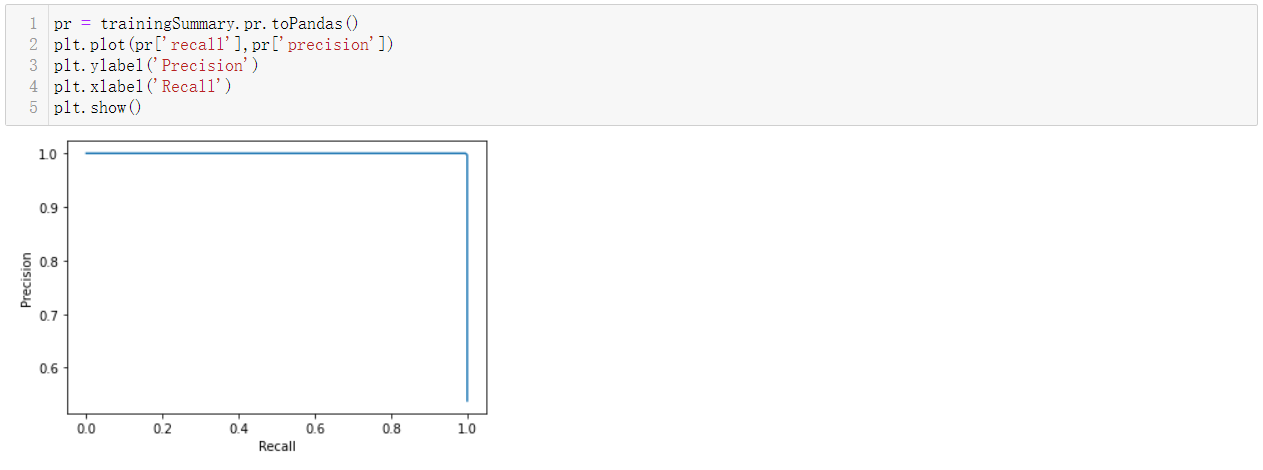
逻辑回归

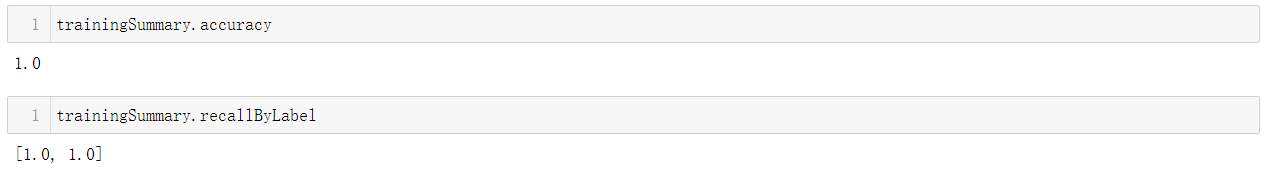




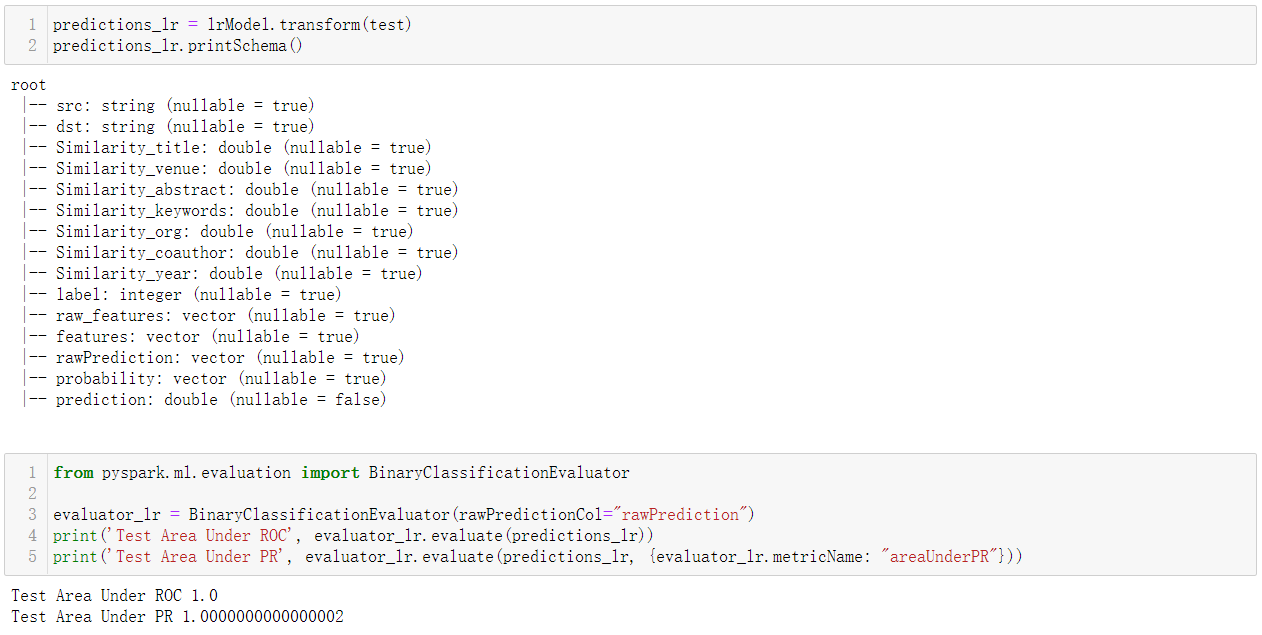
先评估逻辑回归模型在训练集上的性能



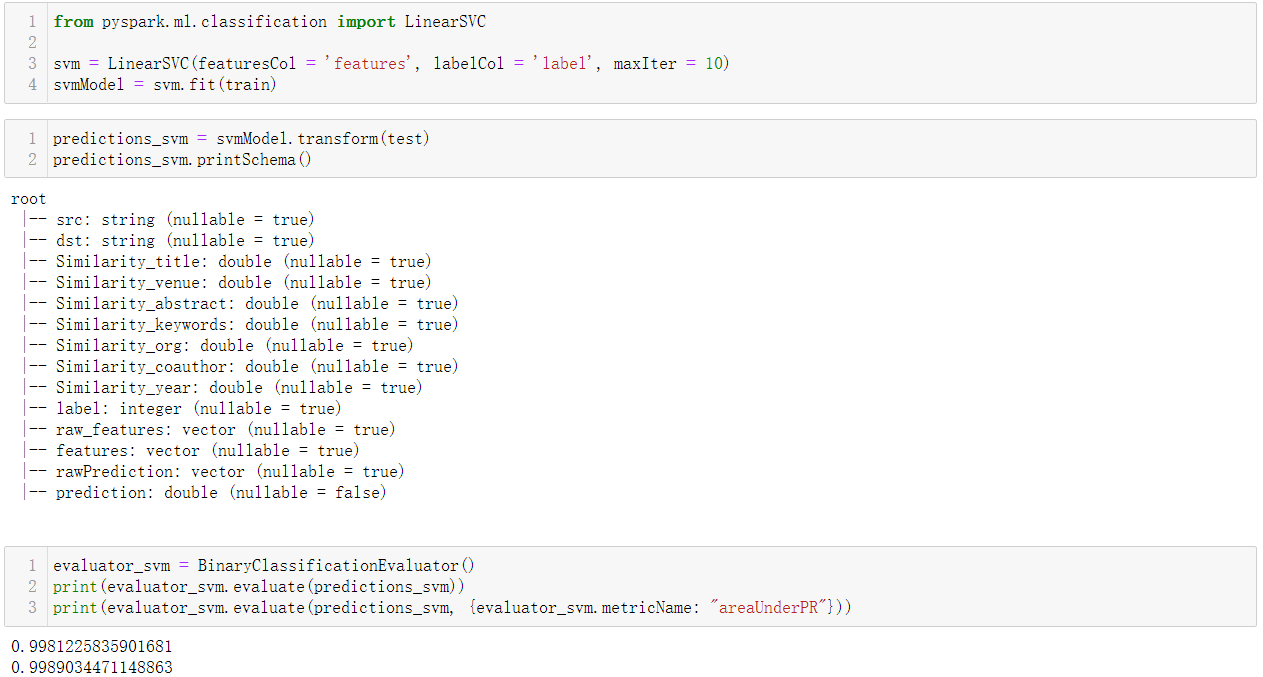




然后评估逻辑回归模型在手动划分的测试集上的性能



SVM

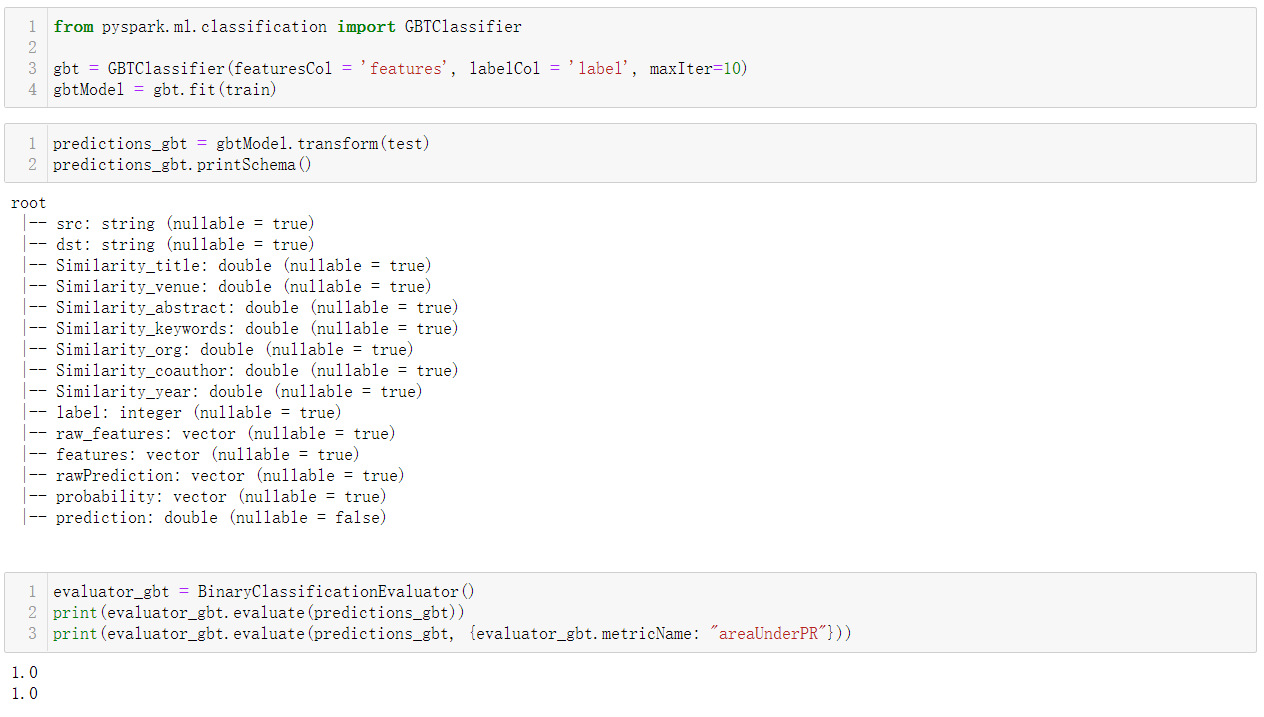


决策树



随机森林

 梯度提升树

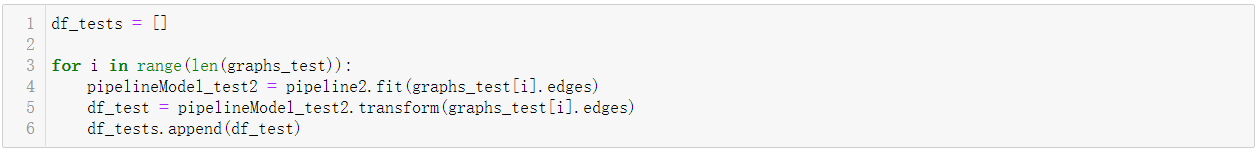


之后评估分类器在真正的测试集上的性能

由于建立全连接图O()的开销，所以同样选择重名论文数量最少的20个名字作为测试集。

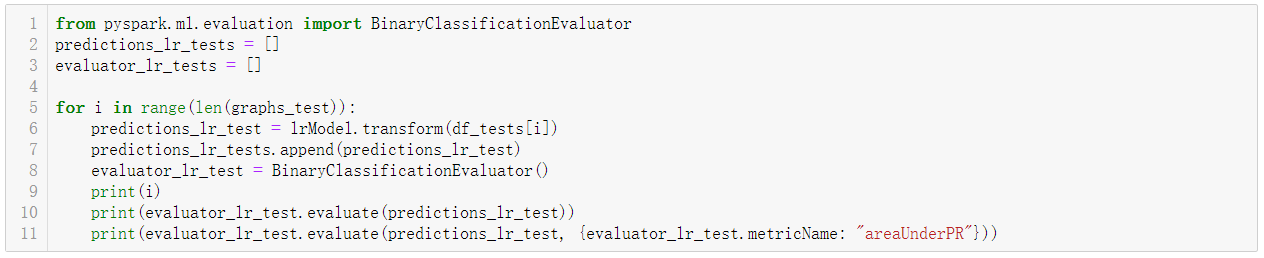
 同样经过之前训练集经过的两个pipeline，然后建立全连接图。

由于要对每个作者单独评估性能，所以用一个列表append所有图的边。



评估分类器在所有测试集图上的性能

逻辑回归



0

0.999875260360926

0.9995952786992442

1

0.9998754143707054

0.9995957911832626

2

0.999876425568048

0.9995992936505276

3

0.9998769156755228

0.9996012409883488

4

0.9998757293086092

0.9995965876381214

5

0.9998757293086091

0.9995965876381209

6

0.9998735394564467

0.9995880738083737

7

0.9998751004984656

0.9995940206815981

8

0.9998759267534022

0.9995975273931816

9

0.9998750993325565

0.9995946272155687

10

0.9998733314466162

0.9995899937057117

11

0.9998672460225858

0.999573581642882

12

0.9998641712037839

0.9995637396162779

13

0.9998622239023388

0.9995608820481607

14

0.9998557789246815

0.9995553441393478

15

0.9998483732270623

0.9995468656273271

16

0.9998357386248997

0.9995407433153987

17

0.9997560371304374

0.9995338652723523

18

0.9998898797539041

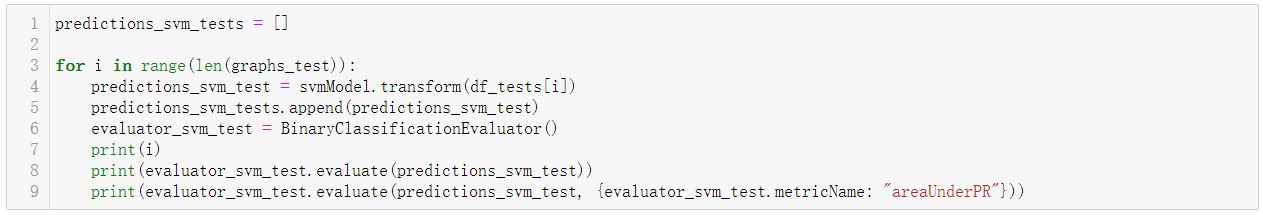
0.9998126406733998

19

0.9998828086869866

0.9998127695548842

SVM



0

0.9795790909683143

0.9736836527210977

1

0.9794637647333464

0.9734850013288512

2

0.9793976398706667

0.9734020524472488

3

0.9790639811999586

0.9729364065281664

4

0.9788799795265989

0.972668816367466

5

0.978879979526603

0.972668816367466

6

0.9786808133905357

0.9723503099655234

7

0.978495449449761

0.9721095274135625

8

0.9781908038297776

0.9716910537226783

9

0.978033910263953

0.9714851736782397

10

0.9771644339549632

0.9703139816568569

11

0.9754087401271923

0.9681766799056907

12

0.9751305934374982

0.9678664190663654

13

0.9743714884868703

0.9670448121983003

14

0.9722606552883293

0.9651194931007626

15

0.9704696943185072

0.9637264019192824

16

0.9676259580975941

0.9621305008192987

17

0.9208835405402755

0.9327086271539392

18

0.9038504293386374

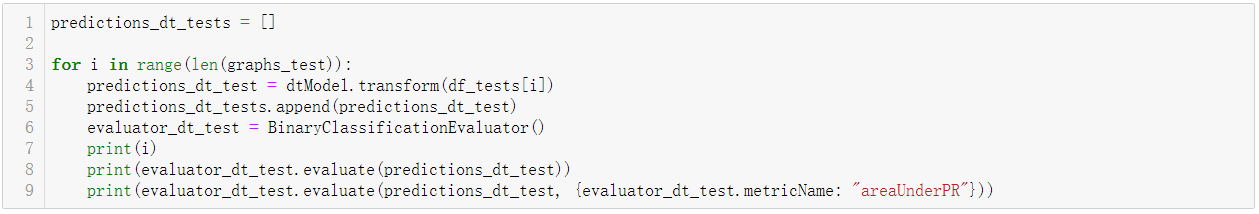
0.9211093154592425

19

0.902887002441993

0.9230827731720911

决策树



0

0.7759132597926738

0.836588529446854

1

0.7787193995394405

0.8384805603957273

2

0.7813893881346011

0.8402855456540708

3

0.7865767172254794

0.8438204033024721

4

0.7873233632427499

0.8442564516322761

5

0.7873233632427499

0.8442564516322761

6

0.7871125972941198

0.8440031300436865

7

0.7899802881882761

0.8460117964258513

8

0.7926876781383807

0.8479157823778869

9

0.7930514605003867

0.8481166886832084

10

0.7970321438334349

0.8509069109950946

11

0.8005679498495032

0.8536018747786003

12

0.7988047809690934

0.8523814670884693

13

0.8006241355285331

0.8539006121195265

14

0.8043822324467506

0.857644058280335

15

0.8049129633533444

0.859128359935402

16

0.8075116951848916

0.8634275764654907

17

0.8142950711895672

0.8836373997190403

18

0.81457399103139

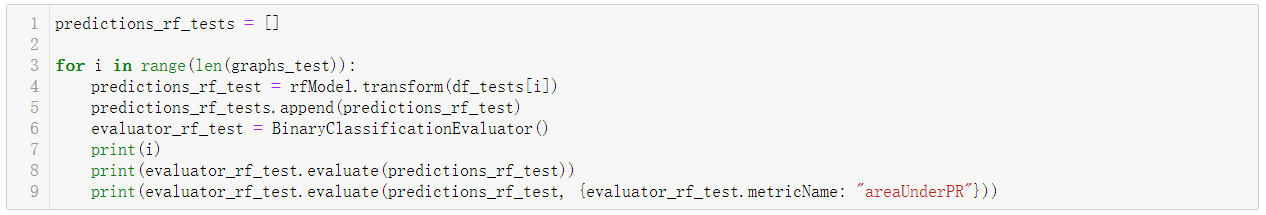
0.8874888208673979

19

0.8144908030506954

0.890223403783296

随机森林



0

0.9986797074754141

0.9981955306131394

1

0.9986630225923748

0.9981769849395797

2

0.9991908068686987

0.9986630363903142

3

0.9991733981660748

0.998641869657327

4

0.9991612930950715

0.9986239510345868

5

0.9991612930950715

0.9986239510345868

6

0.9991492145519506

0.9986013704275152

7

0.9994428028732633

0.9989838843355296

8

0.9994384253977067

0.9989788328070066

9

0.9994316930543827

0.9989659128185511

10

0.9994130727888901

0.9989390348837319

11

0.9997972706904746

0.9993006069765098

12

0.9997924218037292

0.9992832863315417

13

0.9997880895750564

0.9992741075587153

14

0.9997717026974863

0.9992432508471183

15

0.9997565014777213

0.9992160675731956

16

0.9997313313946992

0.9991903289903802

17

0.9995971975547292

0.9991707203976299

18

0.9998119485028208

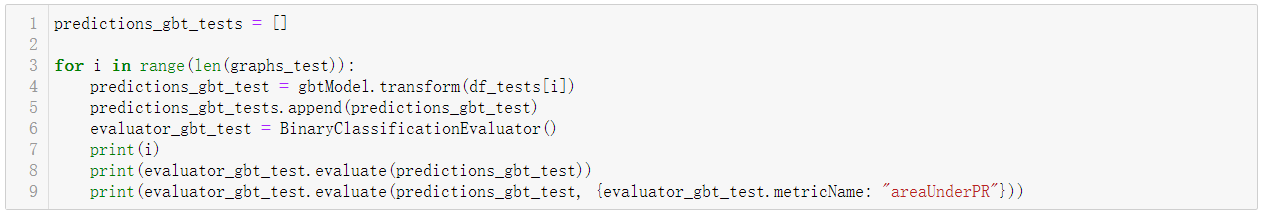
0.9996463150959876

19

0.9997996361792909

0.9996459596015507

梯度提升树



0

0.9981214328565391

0.9979960979284799

1

0.998096017881634

0.9979739399560139

2

0.9984986210540234

0.9983397095120182

3

0.9984567443727581

0.9983045938457016

4

0.998435760711412

0.9982822888221

5

0.998435760711412

0.9982822888221001

6

0.9984119176468924

0.9982567233802115

7

0.9992221921569534

0.9988847981184353

8

0.9992120567960471

0.9988772026715279

9

0.9992003862833766

0.9988625469292878

10

0.9991700759721945

0.9988289572264666

11

0.999806730486342

0.999312757701129

12

0.9998029971121273

0.9992992256082217

13

0.9997988272851588

0.9992905298736869

14

0.999783927260113

0.9992644930902407

15

0.9997691690309034

0.9992369591941695

16

0.9997451323504152

0.9992119266926015

17

0.9996083592546439

0.999162333300652

18

0.9998920951908077

0.9998336467524953

19

0.9998850303232523

0.9998334975232757

计算图的连通分量，并保存结果

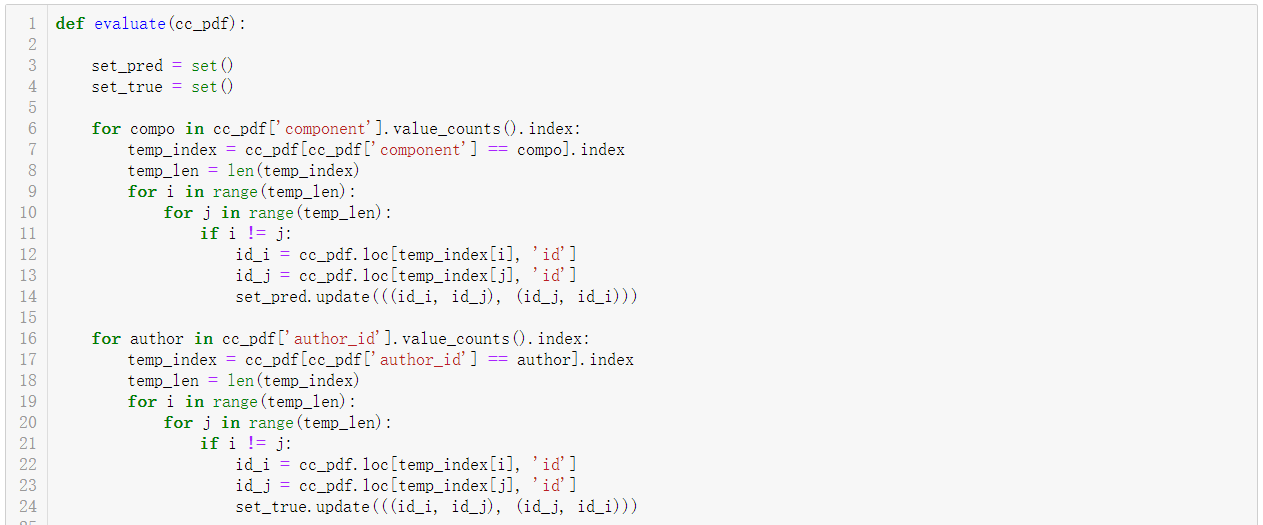


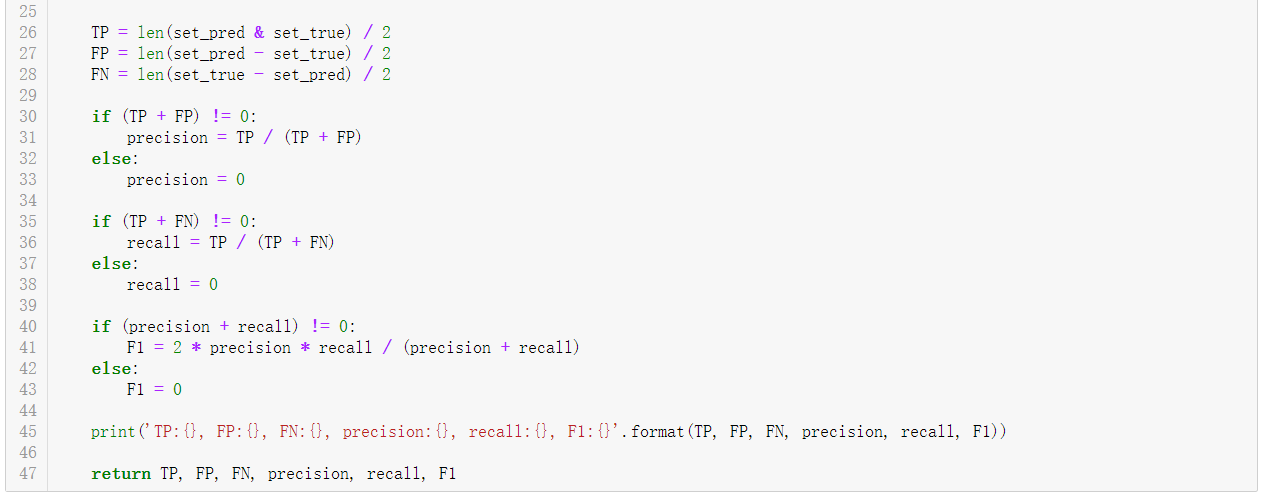
然后计算性能指标

导入保存的连通分量



定义计算各个指标的函数

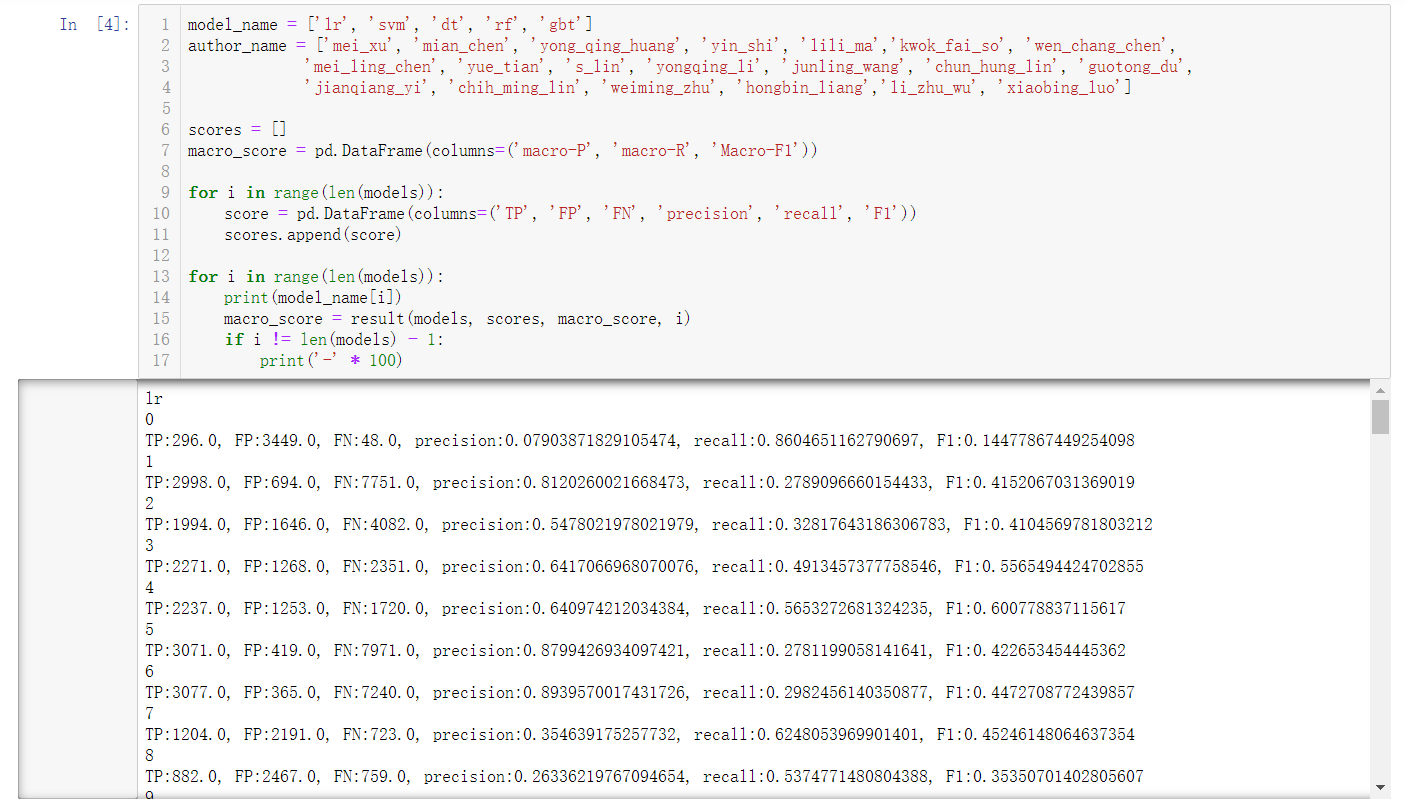


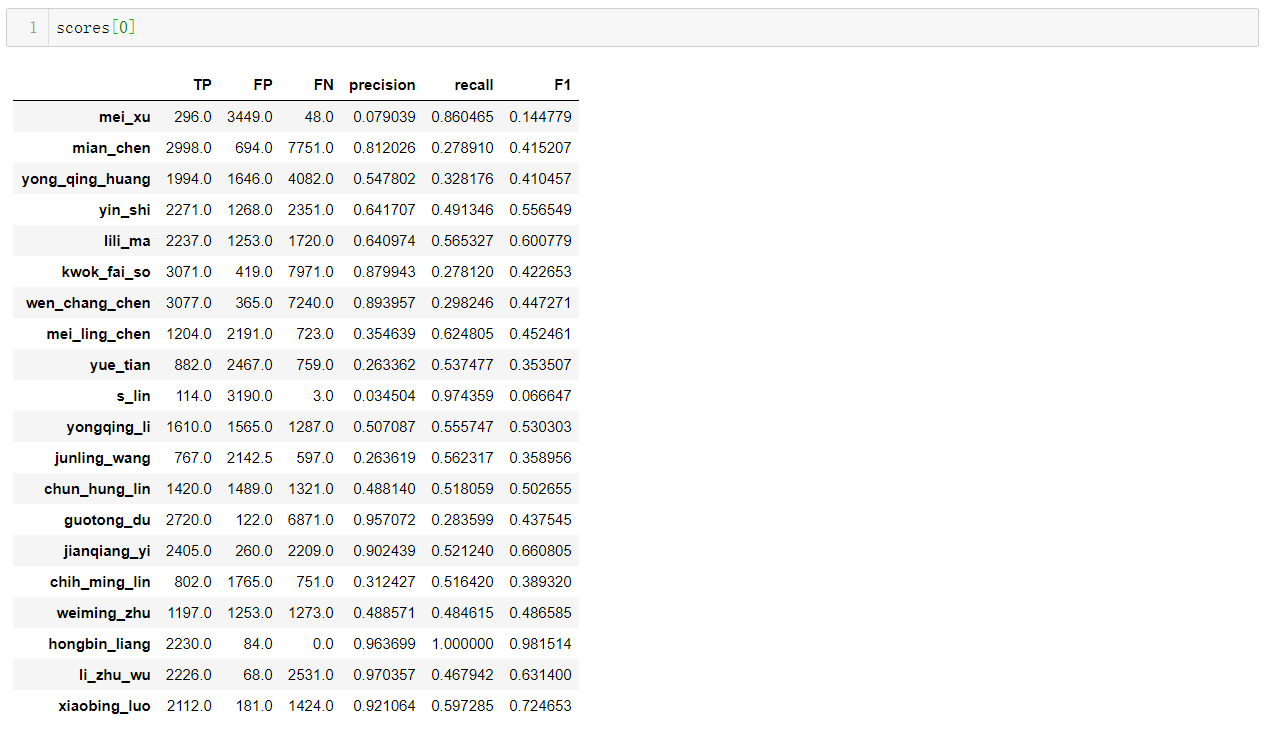


定义函数自动计算五个模型的评估指标



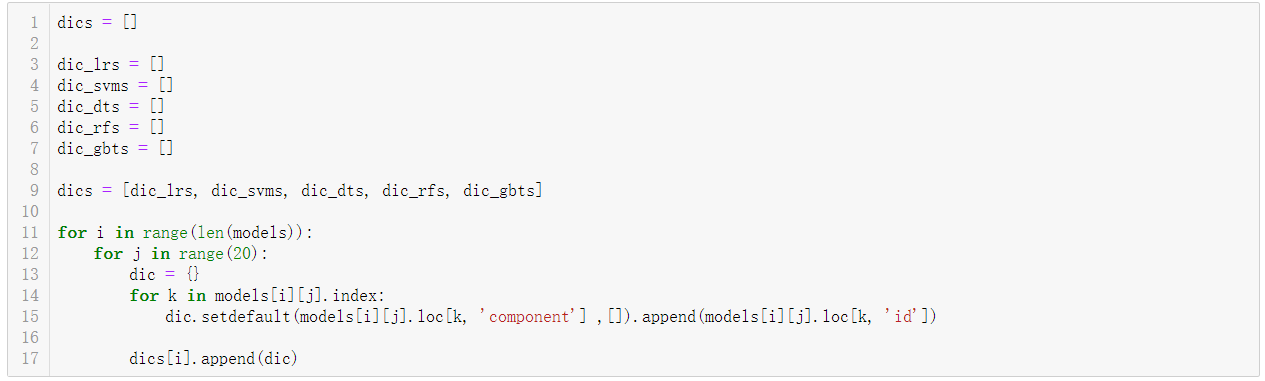
调用上面定义的函数进行评估，并保存评估结果







最后按照output\_example.json的格式保存为json文件





# 4.关键代码说明

## 4.1 从authors列提取出coauthor和org

熟悉python的字典使用。

pdf\_merge**[**'org'**]** **=** **None**

pdf\_merge**[**'coauthor'**]** **=** **None**

pdf\_merge**[**'coauthor\_dict'**]** **=** **None**

**for** i **in** pdf\_merge**.**index**:**

**print(**i**)**

coauthor **=** **[]**

coauthor\_dict **=** **{}**

temp **=** 0

num **=** pdf\_merge**.**loc**[**i**,** 'author\_num'**]**

**for** author **in** pdf\_merge**.**loc**[**i**,** 'authors'**]:**

**if** temp **!=** num**:**

coauthor**.**append**(**author**)**

coauthor\_dict**[**author**[**'name'**]]** **=** author**[**'id'**]**

temp **+=** 1

**if** pdf\_merge**.**loc**[**i**,** 'authors'**][int(**num**)].**get**(**'org'**):**

orgnization **=** pdf\_merge**.**loc**[**i**,** 'authors'**][int(**num**)][**'org'**]**

pdf\_merge**.**loc**[**i**,** 'org'**]** **=** orgnization

pdf\_merge**.**loc**[**i**,** 'coauthor'**]** **=** coauthor

pdf\_merge**.**loc**[**i**,** 'coauthor\_dict'**]** **=** **[**coauthor\_dict**]**

## 4.2建立全连接图

查看文档熟悉graphframes的使用。

**from** graphframes **import** GraphFrame

**from** pyspark**.**sql**.**types **import** StructField**,** StructType**,** StringType**,** IntegerType**,** DoubleType**,** FloatType**,** DecimalType

**import** demjson

**import** time

time\_start**=**time**.**time**()**

graphs **=** **[]**

cnt **=** 0

schema **=** StructType**([**StructField**(**'src'**,** StringType**(),** **True),**

StructField**(**'dst'**,** StringType**(),** **True),**

StructField**(**'Similarity\_title'**,** DoubleType**(),** **True),**

StructField**(**'Similarity\_venue'**,** DoubleType**(),** **True),**

StructField**(**'Similarity\_abstract'**,** DoubleType**(),** **True),**

StructField**(**'Similarity\_keywords'**,** DoubleType**(),** **True),**

StructField**(**'Similarity\_org'**,** DoubleType**(),** **True),**

StructField**(**'Similarity\_coauthor'**,** DoubleType**(),** **True),**

StructField**(**'Similarity\_year'**,** DoubleType**(),** **True),**

StructField**(**'label'**,** IntegerType**(),** **True)])**

**for** name **in** pdf\_clean**[**'name'**].**value\_counts**().**index**:**

vertices **=** df\_clean**[**df\_clean**[**'name'**]** **==** name**].**withColumnRenamed**(**'paper\_id'**,** 'id'**)**

edges\_pdf **=** pd**.**DataFrame**(**columns**=(**'src'**,** 'dst'**,** 'Similarity\_title'**,** 'Similarity\_venue'**,** 'Similarity\_abstract'**,** 'Similarity\_keywords'**,** 'Similarity\_org'**,** 'Similarity\_coauthor'**,** 'Similarity\_year'**,** 'label'**))**

temp\_index **=** pdf\_clean**[**pdf\_clean**[**'name'**]** **==** name**].**index

temp\_len **=** **len(**temp\_index**)**

**for** i **in** **range(**temp\_len**-**1**):**

src **=** pdf\_clean**.**loc**[**temp\_index**[**i**],** 'paper\_id'**]**

**for** j **in** **range(**i**+**1**,** temp\_len**):**

label **=** 0

**if** pdf\_clean**[**'author\_id'**][**i**]** **==** pdf\_clean**[**'author\_id'**][**j**]:**

label **=** 1

dst **=** pdf\_clean**.**loc**[**temp\_index**[**j**],** 'paper\_id'**]**

Similarity\_title **=** Cosine**(**pdf\_clean**[**'title\_word2vec'**][**i**],** pdf\_clean**[**'title\_word2vec'**][**j**])**

Similarity\_venue **=** Cosine**(**pdf\_clean**[**'venue\_word2vec'**][**i**],** pdf\_clean**[**'venue\_word2vec'**][**j**])**

Similarity\_abstract **=** Cosine**(**pdf\_clean**[**'abstract\_word2vec'**][**i**],** pdf\_clean**[**'abstract\_word2vec'**][**j**])**

Similarity\_keywords **=** Cosine**(**pdf\_clean**[**'keywords\_word2vec'**][**i**],** pdf\_clean**[**'keywords\_word2vec'**][**j**])**

Similarity\_org **=** Cosine**(**pdf\_clean**[**'org\_word2vec'**][**i**],** pdf\_clean**[**'org\_word2vec'**][**j**])**

coauthor\_dict\_i **=** demjson**.**decode**(**pdf\_clean**[**'coauthor\_dict'**][**i**])[**0**]**

coauthor\_dict\_j **=** demjson**.**decode**(**pdf\_clean**[**'coauthor\_dict'**][**j**])[**0**]**

Similarity\_coauthor **=** Jaccard**(**coauthor\_dict\_i**,** coauthor\_dict\_j**)**

Similarity\_year **=** **pow(**pdf\_clean**[**'year'**][**i**]** **-** pdf\_clean**[**'year'**][**j**],** 2**)**

edges\_pdf **=** edges\_pdf**.**append**(**pd**.**DataFrame**({**'src'**:[**src**],** 'dst'**:[**dst**],** 'Similarity\_title'**:[**Similarity\_title**],** 'Similarity\_venue'**:[**Similarity\_venue**],** 'Similarity\_abstract'**:[**Similarity\_abstract**],** 'Similarity\_keywords'**:[**Similarity\_keywords**],** 'Similarity\_org'**:[**Similarity\_org**],** 'Similarity\_coauthor'**:[**Similarity\_coauthor**],** 'Similarity\_year'**:[**Similarity\_year**],** 'label'**:[**label**]}),** ignore\_index**=True)**

**print(**cnt**)**

cnt **+=** 1

edges **=** spark**.**createDataFrame**(**edges\_pdf**,** schema**)**

graph **=** GraphFrame**(**vertices**,** edges**)**

graphs**.**append**(**graph**)**

time\_end**=**time**.**time**()**

**print(**'totally cost'**,**time\_end **-** time\_start**)**

## 4.3计算评估指标

利用python集合的交集和并集很容易计算出TP，FP，FN指标。

**def** evaluate**(**cc\_pdf**):**

set\_pred **=** **set()**

set\_true **=** **set()**

**for** compo **in** cc\_pdf**[**'component'**].**value\_counts**().**index**:**

temp\_index **=** cc\_pdf**[**cc\_pdf**[**'component'**]** **==** compo**].**index

temp\_len **=** **len(**temp\_index**)**

**for** i **in** **range(**temp\_len**):**

**for** j **in** **range(**temp\_len**):**

**if** i **!=** j**:**

id\_i **=** cc\_pdf**.**loc**[**temp\_index**[**i**],** 'id'**]**

id\_j **=** cc\_pdf**.**loc**[**temp\_index**[**j**],** 'id'**]**

set\_pred**.**update**(((**id\_i**,** id\_j**),** **(**id\_j**,** id\_i**)))**

**for** author **in** cc\_pdf**[**'author\_id'**].**value\_counts**().**index**:**

temp\_index **=** cc\_pdf**[**cc\_pdf**[**'author\_id'**]** **==** author**].**index

temp\_len **=** **len(**temp\_index**)**

**for** i **in** **range(**temp\_len**):**

**for** j **in** **range(**temp\_len**):**

**if** i **!=** j**:**

id\_i **=** cc\_pdf**.**loc**[**temp\_index**[**i**],** 'id'**]**

id\_j **=** cc\_pdf**.**loc**[**temp\_index**[**j**],** 'id'**]**

set\_true**.**update**(((**id\_i**,** id\_j**),** **(**id\_j**,** id\_i**)))**

TP **=** **len(**set\_pred **&** set\_true**)** **/** 2

FP **=** **len(**set\_pred **-** set\_true**)** **/** 2

FN **=** **len(**set\_true **-** set\_pred**)** **/** 2

**if** **(**TP **+** FP**)** **!=** 0**:**

precision **=** TP **/** **(**TP **+** FP**)**

**else:**

precision **=** 0

**if** **(**TP **+** FN**)** **!=** 0**:**

recall **=** TP **/** **(**TP **+** FN**)**

**else:**

recall **=** 0

**if** **(**precision **+** recall**)** **!=** 0**:**

F1 **=** 2 **\*** precision **\*** recall **/** **(**precision **+** recall**)**

**else:**

F1 **=** 0

**print(**'TP:{}, FP:{}, FN:{}, precision:{}, recall:{}, F1:{}'**.format(**TP**,** FP**,** FN**,** precision**,** recall**,** F1**))**

**return** TP**,** FP**,** FN**,** precision**,** recall**,** F1

# 5.实验总结