

信息检索比赛报告

比赛详情

这次比赛，因为赛道一和赛道二都为论文消歧任务，我做了赛道一和赛道二。其中赛道一排名最好，排名为40。

赛道一

赛道一，我跑了一个模型，在验证集上得分较低，得分为0.304，但在测试集上效果上升，最终得分为0.43454，排名为40。

赛道二

赛道二，我跑了两个模型，花了大部分时间在赛道二上。第一个模型，在验证集上得分为0.64，在测试集上得分为0.598。第二个模型在测试评分较高，结果得分为0.62143，排名为87。

比赛内容

背景介绍

在许多应用中，同名消歧 (Name Disambiguation - aiming at disambiguating WholsWho) 一直被视为一个具有挑战性的问题，如科学文献管理、人物搜索、社交网络分析等，同时，随着科学文献的大量增长，使得该问题的解决变得愈加困难与紧迫。尽管同名消歧已经在学术界和工业界被大量研究，但由于数据的杂乱以及同名情景十分复杂，导致该问题仍未能很好解决。

问题描述

收录各种论文的线上学术搜索系统(例Google Scholar, DbIp和AMiner等)已经成为目前全球学术界重要且最受欢迎的学术交流以及论文搜索平台。然而由于论文分配算法的局限性，现有的学术系统内部存在着大量的论文分配错误；此外，每天都会有大量新论文进入系统。故如何准确快速的将论文分配到系统中已有作者档案以及维护作者档案的一致性，是现有的线上学术系统亟待解决的难题。

由于学术系统内部的数据十分巨大（AMiner大约有130, 000, 000作者档案，以及超过200, 000, 000篇论文），导致作者同名情景十分复杂，要快速且准确的解决同名消歧问题还是有很大的障碍。[1]

竞赛希望提出一种解决问题的模型，可以根据论文的详细信息以及作者与论文之间的联系，去区分属于不同作者的同名论文，获得良好的论文消歧结果。而良好的消歧结果是确保学术系统中，专家知识搜索有效性、数字图书馆的高质量内容管理以及个性化学术服务的重要前提，也可影响到其他相关领域。

赛题描述

1. 论文的冷启动消歧

任务描述：给定一堆拥有同名作者的论文，要求返回一组论文聚类，使得一个聚类内部的论文都是一个人的，不同聚类间的论文不属于一个人。最终目的是识别出哪些同名作者的论文属于同一个人。

参考方法：解决这一问题的常用思路就是通过聚类算法，提取论文特征，定义聚类相似度度量，从而将一堆论文聚成的几类论文，使得聚类内部论文尽可能相似，而类间论文有较大不同，最终可以将每一类论文看成属于同一个人的论文。[7] 是一篇经典的使用聚类方法的论文，它使用了原子聚类的思想，大致思路是首先用较强的规则进行聚类，例如：两篇论文如果有两个以上的共同作者，那么这两篇论文属于同一类，这样可以保证聚类内部的准确率，随后用弱规则将先前的聚类合并，从而提高召回率。有些工作考虑了传统特征的局限性，所以利用了低维语义空间的向量表示方法，通过将论文映射成低维空间的向量表示，从而基于向量使用聚类方法 [2]。

II. 论文的增量消歧

任务描述：线上系统每天会新增大量的论文，如何准确快速的将论文分配到系统中已有作者档案，这是线上学术系统最亟待解决的问题。所以问题抽象定义为：给定一批新增论文以及系统已有的作者论文集，最终目的是把新增论文分配到正确的作者档案中。

参考方法：增量消歧任务与冷启动消歧的任务不同，它是基于有一定作者档案的基础，对新增论文进行分配。所以，容易直接想到的方法就是将已有的作者档案与新增论文进行比较，提取合作者，单位机构或者会议期刊之间相似度的传统特征，随后利用svm之类的传统分类器进行分类。还可以利用基于低维空间的向量表示方法，通过将作者与论文表示成低维向量，使用监督学习方法进行特征提取及模型训练。

评估方法：使用WeightedF1 作为模型评估度量。

数据集

赛道一

数据集：

sna_test_data

train

sna_data

训练集

train_author.json

数据格式：此文件中的数据组织成一个字典（dictionary，记为dic1），存储为JSON对象。dic1的键（key）是作者姓名。dic1的值（value）是表示同名作者集合的字典（记为dic2）。dic2的键（key）是作者ID，dic2的值（value）是该作者的论文ID列表。

train_pub.json：此文件包含train_author.json所有论文的元数据，数据存储为JSON对象；

数据格式：此文件的数据表示为一个字典（dictionary），其键（key）是论文ID，其值是相应的论文信息。

验证集

sn_valid_author_raw.json：二级字典，key值为作者姓名，value为一个论文的list，代表该作者姓名下所有同名作者的论文，参赛者需要将同名作者的论文聚成不同的类簇。

sna_valid_example_evaluation_scratch.json: 示例提交文件，组织成二级字典格式，key值为作者姓名，value值是一个二维列表，第一维的长度代表类簇的数目，第二维代表各个类簇的论文ids。

sna_valid_pub.json: 二级字典，代表验证集所有论文的元信息，格式同train_pub.json。

测试集

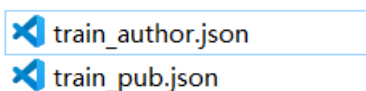
sna_test_author_raw.json: 二级字典，key值为作者姓名，value为一个论文的list，代表该作者姓名下所有同名作者的论文, 参赛者需要将同名作者的论文聚成不同的类簇。

sna_test_example_evaluation_scratch.json: 示例提交文件，组织成二级字典格式，key值为作者姓名，value值是一个二维列表，第一维的长度代表类簇的数目，第二维代表各个类簇的论文ids。

sna_test_pub.json: 二级字典，代表验证集所有论文的元信息，格式同train_pub.json。

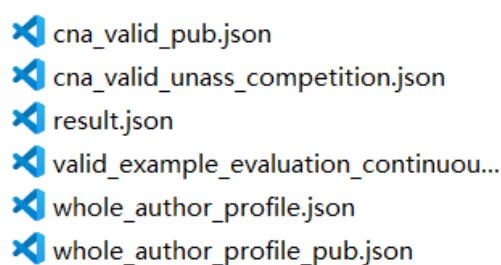
赛道二

训练数据集:



train_author.json
train_pub.json

验证数据集和完整数据集:



cna_valid_pub.json
cna_valid_unass_competition.json
result.json
valid_example_evaluation_continuou...
whole_author_profile.json
whole_author_profile_pub.json

测试数据集:

名称
cna_test_pub.json
cna_test_unass_competition.json
test_example_evaluation_continuous...

训练集

train_author.json

数据格式: 此文件中的数据组织成一个字典 (dictionary, 记为dic1), 存储为JSON对象。dic1的键 (key) 是作者姓名。dic1的值 (value) 是表示作者的字典 (记为dic2)。dic2的键 (key) 是作者ID, dic2的值 (value) 是该作者的论文ID列表。

train_pub.json

此文件包含train_author.json所有论文的元数据, 数据存储为JSON对象;

数据格式：此文件的数据表示为一个字典（dictionary），其键（key）是论文ID，其值是相应的论文信息。

whole_author_profile.json

二级字典，key值为作者id，value分为两个域：'name'域代表作者名，'papers'域代表作者的所拥有的论文(作者的profile)，测试集与验证集使用同一个已有的作者档案；

whole_author_profile_pub.json

whole_author_profile.json中涉及的论文元信息，格式同train_pub.json；

验证集

cna_valid_unass_competition.json

论文列表，代表待分配的论文list，列表中的元素为论文id + '-' + 需要分配的作者index(从0开始)；参赛者需要将该文件中的每篇论文的待分配作者对应分配到已有作者档案中(whole_author_profile.json)

valid_example_evaluation_continuous.json

示例提交文件。二级字典，key值为作者 ID，value 值代表分配到该作者的论文id（来自cna_valid_unass_competition.json）。

cna_valid_pub.json

cna_valid_unass_competition.json中所涉及的论文元信息，格式同train_pub.json。

测试集

cna_test_unass_competition.json

论文列表，代表待分配的论文list，列表中的元素为论文id + '-' + 需要分配的作者index(从0开始)；参赛者需要将该文件中的每篇论文的待分配作者对应分配到已有作者档案中(whole_author_profile.json)。

test_example_evaluation_continuous.json

示例提交文件。二级字典，key值为作者 ID，value 值代表分配到该作者的论文id（来自cna_valid_unass_competition.json）。

cna_test_pub.json

cna_test_unass_competition.json中所涉及的论文元信息，格式同 train_pub.json。

论文知识

看了唐杰老师的论文阅读论文：

ArnetMiner: Extraction and Mining of Academic Social Networks

A Unified Probabilistic Framework for Name Disambiguation in Digital Library

模型建立

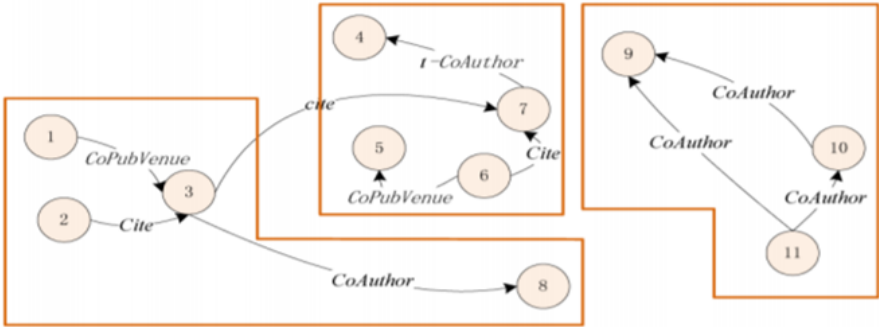
提取论文信属性

Attribute	Description
$p_i.title$	title of p_i
$p_i.pubvenue$	published conference/journal of p_i
$p_i.year$	published year of p_i
$p_i.abstract$	abstract of p_i
$p_i.authors$	authors name set of p_i $\{a_i^{(0)}, a_i^{(1)}, \dots, a_i^{(w)}\}$
$p_i.references$	references of p_i

提取论文之间关系

R	W	Relation Name	Description
r_1	w_1	CoPubVenue	$p_i.pubvenue = p_j.pubvenue$
r_2	w_2	CoAuthor	$\exists r, s>0, a_i^{(r)}=a_j^{(s)}$
r_3	w_3	Citation	p_i cites p_j or p_j cites p_i
r_4	w_4	Constraint	feedback supplied by users
r_5	w_5	τ -CoAuthor	τ -extension co-authorship ($\tau>1$)

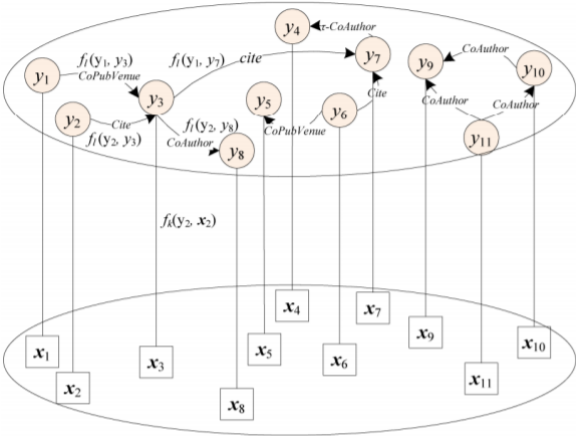
论文的关系图



算法过程

使用了 隐性马尔可夫随机场

Hidden Markov Random Fields



公式：

$$P(Y) = \frac{1}{Z_1} \exp \left(\sum_{(y_i, y_j) \in E, k} \lambda_k f_k(y_i, y_j) \right),$$

$$Z_1 = \sum_{y_i, y_j} \sum_{(y_i, y_j) \in E, k} \lambda_k f_k(y_i, y_j)$$

$$P(X|Y) = \frac{1}{Z_2} \exp \left(\sum_{x_i \in X, l} \alpha_l f_l(y_i, x_i) \right),$$

$$Z_2 = \sum_{y_i} \sum_{x_i \in X, l} \alpha_l f_l(y_i, x_i),$$

使用贝叶斯定理，后验概率转为先验概率

$$L_{\max} = \log(P(Y|X)) = \log(P(Y)P(X|Y)). \quad (4)$$

By substituting (2) and (3) into (4), we obtain

$$L_{\max} = \log \left(\frac{1}{Z_1 Z_2} \exp \left(\sum_{(y_i, y_j) \in E, k} \lambda_k f_k(y_i, y_j) + \sum_{x_i \in X, l} \alpha_l f_l(y_i, x_i) \right) \right).$$

而

$$f_k(y_i, y_j) = K(x_i, x_j) \sum_{r_m \in R_{ij}} [w_m r_m(x_i, x_j)].$$

进行计算，得出 L_{\max} 既可。

比赛算法

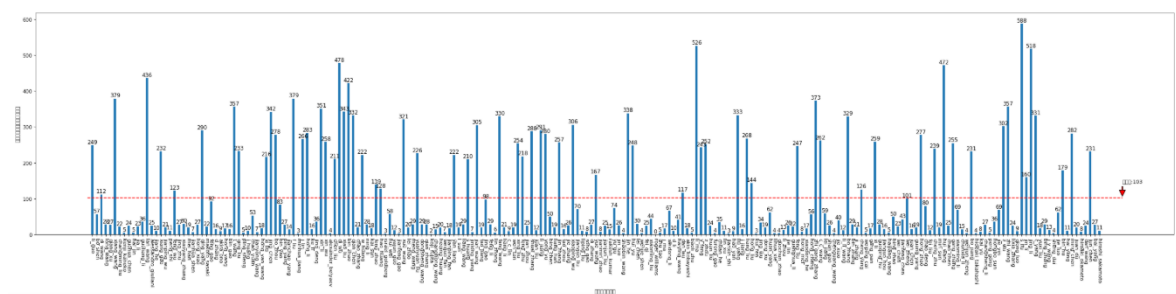
赛道一

数据处理

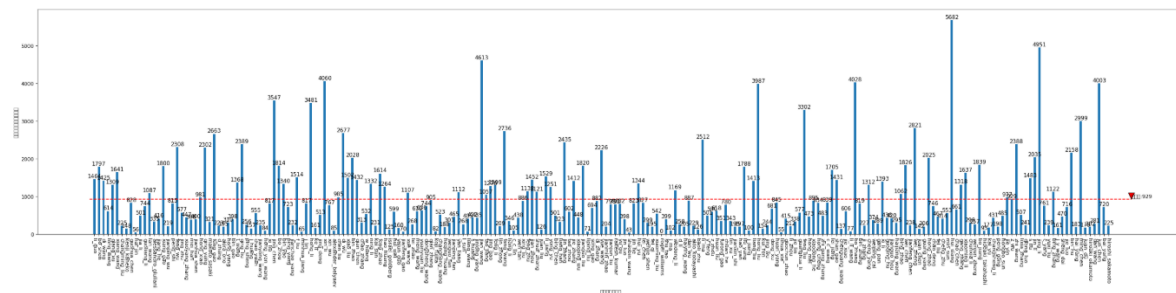
```
authors = [author for author in train_data]
authors_num_person = [len(train_data[author].keys()) for author in train_data]
print('训练集同名数量: ', len(authors))
print('消歧后实际作者数量: ', sum(authors_num_person))
```

训练集同名数量： 221
消歧后实际作者数量： 22839

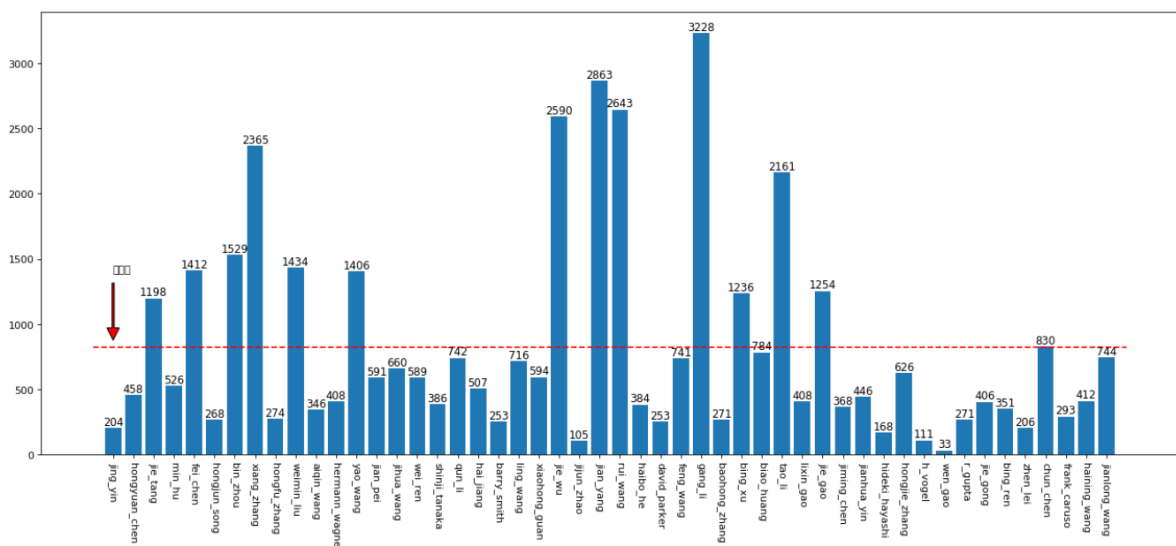
绘制训练集同名作者个体数量



绘制训练集同名作者论文总数



验证集数据分析



查看论文作者名中是否包含消歧作者名

'''

作者名存在不一致的情况：

- 1、大小写
- 2、姓、名顺序不一致
- 3、下划线、横线
- 4、简写与不简写
- 5、姓名有三个字的表达：名字是否分开

同理：机构的表达也存在不一致的情况

因此：需要对数据做相应的预处理统一表达

'''

使用正则去标点

```
#正则去标点
def etl(content):
    content = re.sub("[\s+\.!\/,;$$^*(+\"'\")+|[\+—()?【】“”!，。?、~@#¥%.....&*（）]+", " ", content)
    content = re.sub(r" {2,}", " ", content)
    return content
```

特征提取

1. 基于规则（按组织机构消歧）

```
def disambiguate_by_org():
    res_dict = {}
    for author in validate_data:
        res_dict[author] = []
        print(author)
        papers = validate_data[author]
        org_dict = {}
        org_dict_coauthor = {}
        no_org_list = []
        for paper in papers:
            authors = paper['authors']
            for paper_author in authors:
                name = preprocessname(paper_author['name'])
                org = preprocessorg(paper_author['org']) if 'org' in paper_author else ""
                name = name.split('_')
                if '_' .join(name) == author or '_' .join(name[:-1]) == author:
                    if org == "":
                        no_org_list.append((paper['id'],
                        [preprocessname(paper_author['name']) for paper_author in authors]))
                    else: # 按组织聚类
                        if org not in org_dict:
                            org_dict[org] = [paper['id']]
                            org_dict_coauthor[org] =
                            [preprocessname(paper_author['name']) for paper_author in authors]
                        else:
                            org_dict[org].append(paper['id'])

            org_dict_coauthor[org].extend([preprocessname(paper_author['name']) for
            paper_author in authors])

        # 没有组织的根据合作者交集
        for p_id, names in no_org_list:
            tmp = ""
            max_num = 1
            for org in org_dict_coauthor:
                set_a = set(names)
                set_b = set(org_dict_coauthor[org])
                intersection = set_a & set_b
                iou = len(intersection)
                if iou > max_num:
                    max_num = iou
                    tmp = org
            if max_num != 1:
                org_dict[tmp].append(p_id)
```



```

        else:
            res_dict[author].append([p_id])
    for org in org_dict:
        res_dict[author].append(org_dict[org])
    json.dump(res_dict, open('disambiguate_by_org_result.json', 'w',
encoding='utf-8'), indent=4)
disambiguate_by_org()

```

2. 基于规则（按合作者与组织机构消歧）

```

def disambiguate_by_coauthor():
    res_dict = {}
    for author in validate_data:
        print(author)
        res = []
        papers = validate_data[author]
        print(len(papers))
        paper_dict = {}
        for paper in papers:
            d = {}
            authors = [preprocessname(paper_author['name']) for paper_author in
paper['authors']]
            if author in authors:
                authors.remove(author)
            org = preprocessorg(get_org(paper['authors'], author))
            venue = paper['venue']
            d["authors"] = authors
            d["org"] = org
            d['keywords'] = paper['keywords'] if 'keywords' in paper else ""
            d['venue'] = venue

            if len(res) == 0:
                res.append([paper['id']])
            else:
                max_inter = 0
                indx = 0
                for i, clusters in enumerate(res):
                    score = 0
                    for pid in clusters:
                        insection = set(paper_dict[pid]['authors']) &
set(authors)

                        score += len(insection)
                    if score > max_inter:
                        max_inter = score
                        indx = i
                if max_inter > 0:
                    res[indx].append(paper['id']) # 若最高分大于0，将id添加到得分最高
的簇中

            else:
                res.append([paper['id']]) # 否则，另起一簇
                paper_dict[paper['id']] = d
        res_dict[author] = res
    json.dump(res_dict, open('disambiguate_by_coauthor_result.json', 'w',
encoding='utf-8'), indent=4)
disambiguate_by_coauthor()

```

聚类方法

使用 无监督聚类（根据合作者和机构TFIDF进行相似度聚类），使用DBSCAN作为聚类方法，然后进行调参将 eps, min_samples参数 进行调参

```
def disambiguate_by_cluster():
    res_dict = {}
    for author in validate_data:
        print(author)
        coauthor_orgs = []
        papers = validate_data[author]
        if len(papers) == 0:
            res_dict[author] = []
            continue
        print(len(papers))
        paper_dict = {}
        for paper in papers:
            authors = paper['authors']
            names = [preprocessname(paper_author['name']) for paper_author in
authors]
            orgs = [preprocessorg(paper_author['org']) for paper_author in
authors if 'org' in paper_author]
            abstract = paper['abstract'] if 'abstract' in paper else ''
            coauthor_orgs.append(etl(' '.join(names + orgs) + ' ' + abstract))

        tfidf = TfidfVectorizer().fit_transform(coauthor_orgs)
        clf = DBSCAN(eps=0.58188, min_samples=1, metric='cosine',
algorithm='auto', leaf_size=30, p=None, n_jobs=1)
        s = clf.fit_predict(tfidf)
        #每个样本所属的簇
        for label, paper in zip(clf.labels_, papers):
            if str(label) not in paper_dict:
                paper_dict[str(label)] = [paper['id']]
            else:
                paper_dict[str(label)].append(paper['id'])
        res_dict[author] = list(paper_dict.values())
    json.dump(res_dict, open('disambiguate_by_cluster_result.json', 'w',
encoding='utf-8'), indent=4)
disambiguate_by_cluster()
```

结果预测

0.432123533242837	Success	disambiguate_by_cluster_result.json 2019年12月2日 13:51
0.433877139408719	Success	disambiguate_by_cluster_result.json 2019年12月2日 13:58
0.432244764934408	Success	disambiguate_by_cluster_result3.json 2019年12月2日 14:08
0.36614039002137	Success	disambiguate_by_cluster_result4.json 2019年12月2日 14:13
0.434543311180903	Success	disambiguate_by_cluster_result5.json 2019年12月2日 14:16
0.432986290711888	Success	disambiguate_by_cluster_result.json 2019年12月2日 15:18

得到分数为0.4338771

赛道二

模型一

数据处理

读取训练数据，只取同名作者数大于等于5个，作为的名字作为训练集。随机采样500个训练例子，一个训练例子包含paper和正例作者以及5个负例作者（正负例比=1: 5）

```
# 把采样的500篇paper转变成对应的训练例子，一个训练例子包含paper和正例作者以及5个负例作者（正负例比=1: 5）
train_instances = []
for paper_id in train_paper_list:
    # 保存对应的正负例
    pos_ins = set()
    neg_ins = set()
    paper_author_name = paper2aid2name[paper_id][0] # 得到 paper 对应的
author_name
    paper_author_id = paper2aid2name[paper_id][1] # 得到 paper 的 personid
    pos_ins.add((paper_id, paper_author_id)) # 加入 paper_id 对应 personid
    # 获取同名的所有作者(除了本身)作为负例的candidate
    persons = list(author_data[paper_author_name].keys()) # 得到 author_name
的所有 personid
    persons.remove(paper_author_id)
    # 如果该 bool 表达式为 True，该程序可以继续向下执行；否则程序会引发 AssertionError 错误。
    assert len(persons) == (len(list(author_data[paper_author_name].keys())) - 1)

    # 每个正例采样5个负例
    neg_author_list = random.sample(persons, 5)
    for i in neg_author_list:
        neg_ins.add((paper_id, i))
    train_instances.append((pos_ins, neg_ins))
print(len(train_instances)) # 500
```

输出

```
[({'Z1MhIXcc', 'pXj6Fmct'}, {'Z1MhIXcc', 'sVGyDRto'}, {'Z1MhIXcc', 'w5hGoEWM'}, {'Z1MhIXcc', '3VGZ30Ma'}, {'Z1MhIXcc', 'wE5f1v0e'}, {'Z1MhIXcc', 'X7thS4Rj'})), ({('HC9s3JII', 'g9HCfa3s'}, {'HC9s3JII', 'leryEtWM'}, {'HC9s3JII', 'xPslujV0'}, {'HC9s3JII', 'x4qmolFz'}, {'HC9s3JII', '8KjaAATx'}, {'HC9s3JII', 'DNMyLXh6'})), ({('dpBhQ1DJ', 'rZHpLEYM'}, {'dpBhQ1DJ', 'IcriHo7h'}, {'dpBhQ1DJ', 'nryVfLnc'}, {'dpBhQ1DJ', 'PZNoBNcT'}, {'dpBhQ1DJ', 'rv7Zx1fh'}, {'dpBhQ1DJ', 'mAKKG8Ah'})), ({('mkdvvhvY4', '0xyc4cUQ'}, {'mkdvvhvY4', 'Tq8z8fnY'}, {'mkdvvhvY4', '9qer6hQJ'}, {'mkdvvhvY4', '8VRGyfGh'}, {'mkdvvhvY4', 'IjQP560J'}, {'mkdvvhvY4', 'ooYikI7f'})), ({('TA98Ivgq', 'S0KddW6h'}, {'TA98Ivgq', 'kk7noJ6Z'}, {'TA98Ivgq', 'YzWKEQIa'}, {'TA98Ivgq', 'YEZtwY45'}, {'TA98Ivgq', '5202xfkq'}, {'TA98Ivgq', 'tEbedhLg'})), ({('OV1816xf', '8JoCgl61'}, {'OV1816xf', 'YzWKEQIa'}, {'OV1816xf', 'vVFWt6aw'}, {'OV1816xf', 'eEijIIUM'}, {'OV1816xf', 'uaM4EHfa'}, {'OV1816xf', 'WXX6r6Jz'})), ({('VKyWD6sQ', 'XxSeEGCL'}, {'VKyWD6sQ', '00F2U8bv'}, {'VKyWD6sQ', 'vt6pModK'}, {'VKyWD6sQ', 'avyhOyug'}, {'VKyWD6sQ', 'XFD0X8yN'}, {'VKyWD6sQ', 'zrRy0iI3'})), ({('IuYvjQuH', 'Mwrm80Qz'}, {'IuYvjQuH', 'T1RwKwmS'}, {'IuYvjQuH', 'Qd9JeKZo'}, {'IuYvjQuH', 'BQXfcBUL'}, {'IuYvjQuH', '8Rmpzouq'}, {'IuYvjQuH', '801b5wiv'})), ({('qYVjxQks', 'T7ZAKFQj'}, {'qYVjxQks', 'rR0sPyyp'}, {'qYVjxQks', 'tvMgxeH8'}, {'qYVjxQks', 'OpLF5rFp'}, {'qYVjxQks', 'OkBf0D71'}, {'qYVjxQks', '53fGvgmQ'})), ({('3cR2T4EM', 'BRkyRGS0'}, {'3cR2T4EM', 'VzSmQQRZ'}, {'3cR2T4EM', 't0ZyTtuj'}, {'3cR2T4EM', 'CwJdy9LJ'}, {'3cR2T4EM', 'iTeiLSWX'}, {'3cR2T4EM', 'uhGo9Vm7'})), ({('TKHdyr1l', 'r7KwXgaX'}, {'TKHdyr1l', '2Hriw5yM'}, {'TKHdyr1l', 'mC3gjPCM'}, {'TKHdyr1l', 'f39mSVYy'}, {'TKHdyr1l', 'i0FODy61'}, {'TKHdyr1l', 'cA4fgvjjs'})), ({('1CH9H3Sr', 'Pd9DQrjf'}, {'1CH9H3Sr', 'B7ogWj0w'}, {'1CH9H3Sr', 'i08T1hFV'}, {'1CH9H3Sr', 'u1
```

对名字的名字进行清洗

```
# 对author_name 进行清洗
def clean_name(name):
    if name is None:
        return ""
    x = [k.strip() for k in name.lower().strip().replace(".", "").replace("-", " ").replace("_", ' ').split()]
    # x = [k.strip() for k in name.lower().strip().replace("-", "").replace("_", ' ').split()]
    full_name = ' '.join(x)
    name_part = full_name.split()
    if(len(name_part) >= 1):
        return full_name
    else:
        return None
```

提取特征

在这里，我们只提取paper与author之间的coauthor相关的特征

```
# 计算了5维paper与author所有的paper的coauthor相关的特征：
# 1. 不重复的coauthor个数
# 2. 不重复的coauthor个数 / paper的所有coauthor的个数
# 3. 不重复的coauthor个数 / author的所有paper不重复coauthor的个数
# 4. coauthor个数（含重复）
# 4. coauthor个数（含重复）/ author的所有paper的coauthor的个数（含重复）
```

分类器

使用SVM分类器进行分类，并对参数进行了调整

```
from sklearn.svm import SVC
from sklearn.externals import joblib

# 构建svm正负例
svm_train_ins = []
for ins in pos_features:
    svm_train_ins.append((ins, 1))

for ins in neg_features:
    svm_train_ins.append((ins, 0))

print(np.array(svm_train_ins).shape)

random.shuffle(svm_train_ins) # 随机排序

x_train= []
y_train = []
for ins in svm_train_ins:
    x_train.append(ins[0])
    y_train.append(ins[1])
```

```
#clf = SVC(probability=True)
clf=SVC(C=2.0,probability=True)
clf.fit(x_train, y_train)
```

结果预测

```
result_dict = defaultdict(list)
for paper_id, ins_feature_list in paper2features.items():
    score_list = []
    for ins in ins_feature_list:
        # 利用svm对一篇paper的所有candidate author去打分，利用分数进行排序，取top-1
        # author作为预测的作者
        prob_pred = clf.predict_proba([ins])[:, 1]
        score_list.append(prob_pred[0])
    rank = np.argsort(-np.array(score_list))
    #取top-1 author作为预测的作者
    predict_author = paper2candidates[paper_id][rank[0]]
    result_dict[predict_author].append(paper_id)
```

验证集上分数

0.622816017740985	weighted precision: 0.7095096513570069 recall: 0.555001447428008 f1: 0.6228160177409852
-------------------	--

测试集上分数

0.598219546789606	weighted precision: 0.6693963503399997 recall: 0.5407244094919996 f1: 0.5982195467896058
-------------------	---

模型二

数据处理

对训练集，提取论文中的信息 abstract、keywords、title、venue、year、authors、orgs，接下来对这些信息进行处理

就是下面这些信息

Attribute	Description
$p_i.title$	title of p_i
$p_i.pubvenue$	published conference/journal of p_i
$p_i.year$	published year of p_i
$p_i.abstract$	abstract of p_i
$p_i.authors$	authors name set of p_i $\{a_i^{(0)}, a_i^{(1)}, \dots, a_i^{(n)}\}$
$p_i.references$	references of p_i

对验证集进行一样的处理

abstract

100% 25911/25911 [00:02<00:00, 10346.23it/s]

keywords

100% 25911/25911 [00:02<00:00, 10445.74it/s]

title

100% 25911/25911 [00:02<00:00, 10592.88it/s]

venue

100% 25911/25911 [00:02<00:00, 10509.94it/s]

year

100% 25911/25911 [00:02<00:00, 10957.82it/s]

authors

100% 25911/25911 [00:02<00:00, 10509.32it/s]

orgs

100% 25911/25911 [00:02<00:00, 10096.91it/s]

对author_name 进行数据清洗

```
def convert(name):  
    name = name.lower()  
    name = name.replace('. ', '_').replace('.', '_').replace(' ',  
    '_').replace('-', '')  
    return name
```

author_org_match 作者单位匹配

将测试集作者名，训练集作者名，完全数据集的作者名求交，看交集

```
print('valid: ', len(set(valid_data['author_name'])))  
print('train: ', len(set(train_author_name_ids['author_name'])))  
print('whole: ', len(set(whole_author_name_paper_ids['author_name'])))  
print('-'*60)  
print('valid & train: ', len(set(train_author_name_ids['author_name']) &  
set(valid_data['author_name'])))  
print('whole & valid: ', len(set(whole_author_name_paper_ids['author_name']) &  
set(valid_data['author_name'])))  
print('whole & train: ', len(set(whole_author_name_paper_ids['author_name']) &  
set(train_author_name_ids['author_name'])))
```

```

valid: 36
train: 221
whole: 320
-----
valid & train: 0
whole & valid: 36
whole & train: 220
2

```

提取特征

gen_feat_v1 构造特征

author_id、author_name、author_org、label、paper_id 这些共有特征，还有 论文A和论文B的属性：abstract、keywords、title、venue、year、authors、orgs 这些信息提取为特征

```

Index(['author_id', 'author_name', 'author_org', 'label', 'paper_id',
      'abstract_a', 'authors_a', 'keywords_a', 'title_a', 'venue_a', 'year_a',
      'orgs_a', 'paper_ids', 'paper_ids_len', 'abstract_b', 'keywords_b',
      'title_b', 'venue_b', 'year_b', 'authors_b', 'orgs_b', 'idx'],
      dtype='object')

```

单独对year特征进行处理，选择 max-year，min-year，mean-year 等特征，然后进行处理

```

cols = ['year_a', 'year_b_min', 'year_b_max', 'year_b_mean', 'year_b_std',
        'year_b_mm2',
        'year_b_min-year_a', 'year_b_max-year_a', 'year_b_mean-year_a',
        'year_b_mm2-year_a', 'year_inside_range']

```

最后还专门提取 组织机构和 作者兴趣这些特征

```

tmp['author_interaset_num'] = tmp.apply(lambda row: func(row['authors_a'],
row['authors_b']), axis=1)

tmp['author_interaset_num/paper_ids_len'] = (tmp['author_interaset_num'] /
(data['paper_ids_len'] - (data['label'] == 1).astype(int))).fillna(0)

tmp.head()

tmp['author_interaset_num'].value_counts()

tmp[['author_org_in_orgs_b_times', 'author_interaset_num',
'author_interaset_num/paper_ids_len']].to_pickle('./feat/tmp.pkl') # 组织机构和 作者兴趣

```

分类器

这次选择的是 Catboost分类器，对数据还进行 pool操作，然后对CatBoostClassifier的参数，比如 iterations，learning_rate等参数值进行了调试，选择了调试效果最优的值

```

X_train, y_train, X_valid, y_valid = trn_dat[used_feat], trn_dat['label'],
val_dat[used_feat], val_dat['label']
cate_features = []
train_pool = Pool(X_train, y_train, cat_features=cate_features)
eval_pool = Pool(X_valid, y_valid, cat_features=cate_features)
if not has_saved:
    cbt_model = CatBoostClassifier(iterations=13000,      # 13000
                                   learning_rate=0.1,
                                   eval_metric='AUC',
                                   use_best_model=True,
                                   random_seed=42,
                                   logging_level='verbose',
                                   task_type='GPU',
                                   devices='0',
                                   gpu_ram_part=0.5,
                                   early_stopping_rounds=300,
                                   loss_function='Logloss',
                                   #
                                   depth=12,
                                   )
    cbt_model.fit(train_pool, eval_set=eval_pool, verbose=100)

```

使用交叉验证法进行训练

```
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

训练模型

```

fold 1 round 549 : auc: 0.959498 | mean auc 0.959498 | F1: 0.936477 | mean F1: 0.936477
*****

```

Warning: less than 75% gpu memory available for training. Free: 4519.5625 Total: 10989.4375

0:	learn: 0.9527896	test: 0.9539095 best: 0.9539095 (0)	total: 32ms	remaining: 6m 56s
100:	learn: 0.9591096	test: 0.9598400 best: 0.9598552 (91)	total: 3.17s	remaining: 6m 44s
200:	learn: 0.9597808	test: 0.9596312 best: 0.9598552 (91)	total: 6.29s	remaining: 6m 40s
300:	learn: 0.9600943	test: 0.9596438 best: 0.9598552 (91)	total: 9.42s	remaining: 6m 37s

bestTest = 0.9598551989

bestIteration = 91

Shrink model to first 92 iterations.

total_unassigned_paper: 49504

true author num: 5502

weighted_precision: 0.939741, weighted_recall: 0.942590, weighted_f1: 0.941164

```

fold 2 round 91 : auc: 0.959855 | mean auc 0.959677 | F1: 0.941164 | mean F1: 0.938821
*****

```

Warning: less than 75% gpu memory available for training. Free: 4519.5625 Total: 10989.4375

0:	learn: 0.9532130	test: 0.9515151 best: 0.9515151 (0)	total: 44.8ms	remaining: 9m 42s
100:	learn: 0.9594922	test: 0.9582524 best: 0.9582717 (95)	total: 3.32s	remaining: 7m 4s
200:	learn: 0.9600809	test: 0.9583285 best: 0.9584846 (166)	total: 6.51s	remaining: 6m 54s
300:	learn: 0.9603662	test: 0.9584463 best: 0.9584876 (253)	total: 9.82s	remaining: 6m 54s
400:	learn: 0.9604937	test: 0.9583033 best: 0.9584876 (253)	total: 13s	remaining: 6m 49s
500:	learn: 0.9607429	test: 0.9583693 best: 0.9584876 (253)	total: 16.2s	remaining: 6m 43s

bestTest = 0.9584876299

bestIteration = 253

Shrink model to first 254 iterations.

total_unassigned_paper: 42833

true author num: 4579

weighted_precision: 0.931870, weighted_recall: 0.929470, weighted_f1: 0.930668

```

fold 3 round 253 : auc: 0.958488 | mean auc 0.959280 | F1: 0.930668 | mean F1: 0.936103
*****

```

weighted_precision: 0.958823, weighted_recall: 0.957028, weighted_f1: 0.957925

```

fold 4 round 802 : auc: 0.986027 | mean auc 0.985487 | F1: 0.957925 | mean F1: 0.959245
*****

```

Warning: less than 75% gpu memory available for training. Free: 4519.5625 Total: 10989.4375

0:	learn: 0.9751739	test: 0.9716462 best: 0.9716462 (0)	total: 13.8ms	remaining: 4m 8s
100:	learn: 0.9856058	test: 0.9828458 best: 0.9828458 (100)	total: 1.27s	remaining: 3m 45s

结果预测

在验证集上测试效果为

0.740733310618693

weighted precision: 0.77119806349501 recall:
0.7125839979720124 f1: 0.7407333106186929

在测试集上测试效果为

0.621181005418131

weighted precision: 0.6782962736939985 recall:
0.5729373788939995 f1: 0.6211810054181308

更换分类器

采用xgboost分类器

```
import xgboost as xgb
X_train, y_train, X_valid, y_valid = trn_dat[used_feat], trn_dat['label'],
val_dat[used_feat], val_dat['label']
dtrain = xgb.DMatrix(X_train,y_train)
dtest = xgb.DMatrix(X_valid)

params = {
    'booster': 'gbtree',
    'objective': 'multi:softmax',
    'num_class': 3,
    'gamma': 0.1,
    'max_depth': 6,
    'lambda': 2,
    'subsample': 0.7,
    'colsample_bytree': 0.7,
    'min_child_weight': 3,
    'silent': 1,
    'eta': 0.1,
    'seed': 1000,
    'nthread': 4,
}

num_round = 500
if not has_saved:
    bst=xgb.train(param, dtrain, num_round)

val_dat['pred'] =bst.predict(dtest)
```

```

*****
total_unassigned_paper: 28493
true author num: 2983
weighted_precision: 0.952579, weighted_recall: 0.950760, weighted_f1: 0.951669
*****
total_unassigned_paper: 49504
true author num: 5502
weighted_precision: 0.954873, weighted_recall: 0.950873, weighted_f1: 0.952869
*****
total_unassigned_paper: 42833
true author num: 4579
weighted_precision: 0.957670, weighted_recall: 0.947961, weighted_f1: 0.952791
*****
total_unassigned_paper: 48473
true author num: 5544
weighted_precision: 0.957212, weighted_recall: 0.954696, weighted_f1: 0.955953
*****
total_unassigned_paper: 36113
true author num: 4213
weighted_precision: 0.957262, weighted_recall: 0.952150, weighted_f1: 0.954699

```

分类器，使用了 xgBoostClassifier 分类器，我进行了调参，选择了比较好的参数。在最终评分上 0.602，没有catboostclassifier 分类器效果好，就没有再继续尝试。

测试集上遇到问题

1、完整数据集和测试集，名字交叉的个数为：65，只有测试集的 1/3，始终没解决这个问题

测试有 186个，交叉的只有65个

```

valid: 186
whole: 320
-----
whole & valid: 65

```

验证集有 36个，全部都交叉了

```

valid: 36
whole: 320
-----
whole & valid: 36

```

分析可能原因测试集中英文名字太多，对名字的预处理的问题，但经过改变，发现还是无法解决，所以需要名字进行模糊匹配，但在提交之前都没有做成，提交完结果之后，经过询问知道了如何进行模糊匹配

可以用下面的方法进行模糊匹配

```

def score(n1, n2):
    n1 = ''.join(filter(str.isalpha, n1.lower()))
    if check_chs(n1):
        # print(n1)
        n1 = to_pinyin(n1)
        # print(n1)
    n2 = ''.join(filter(str.isalpha, n2.lower()))
    counter = defaultdict(int)
    score = 0

```

```
for c in n1:
    counter[c] += 1
for c in n2:
    if (c in counter) and (counter[c] > 0):
        counter[c] -= 1
    else:
        score += 1
score += np.sum(list(counter.values()))
return score
```

结果与分析

1. 对于论文之间进行名字消歧，我们主要靠提取论文的属性，论文之间的关系，提取特征，建立论文关系图，使用聚类或者分类的方法进行名字消歧
2. 对于赛道一，使用聚类方法，提取特征之后，使用DBSCAN方法进行聚类，得出结果
3. 对于赛道二，提取特征，还是多提取一些论文特征，效果越好，分类器使用了 svm，catboost，xgboost分类器，得出catboost效果最好
4. 赛道二最后，还需要专门处理一下名字进行匹配的问题，使用模糊匹配来进行匹配。