地址模糊匹配在电力企业应用中的研究与实践

刘浩宇,李　喆,佘佐超,应卓君,王　薇,赵志浩
(国网四川省电力公司成都供电公司, 四川 成都　610041)

摘　要:地址识别作为大数据应用中自然语言处理的重要场景之一,是重要且极具实用价值的技术手段。目前电力企业正在积极推进大数据应用演进进程,使用大数据技术为电力企业发展赋能的需求也愈加旺盛。而电力企业众多数据资产中的地址信息,作为联系设备—客户—企业的核心字段,其分析挖掘价值极高。首先,针对电力企业常见地址信息涉及的业务应用场景进行分析,瞄准需求要点,对核心应用场景的地址识别需求制定提取匹配方法;然后,基于数据样本对地址识别方法的准确性及运行速度进行研究分析,比对算法的实用性;最后,对算法进行了总结,并对未来算法的提升方式进行展望,以供电力企业相关业务在实际应用中进行参考。

关键词:自然语言处理;模糊匹配;地址识别;电力企业;大数据
中图分类号:TM76　**文献标志码**:A　**文章编号**:1003-6954(2020)06-0043-06

Research and Practice of Address Fuzzy Matching in Application of Power Enterprise

Liu Haoyu, Li Zhe, She Zuochao, Ying Zhuojun, Wang Wei, Zhao Zhihao
(State Grid Chengdu Electric Power Supply Company, Chengdu 610041, Sichuan, China)

Abstract: Address recognition, as one of the important scenarios of natural language processing in big data applications, is an important and highly practical technical means. At present, power enterprises are actively promoting the evolution of big data applications, and the demand of using big data technology to empower the development of power enterprises is also increasing. The address information in many data assets of power enterprises, as the core field of connecting equipment-customer-enterprise, is extremely valuable for analysis and mining. The common address information application scenarios of power enterprises are first analyzed, and aiming at the key points, the extracting and matching methods are formulated for the address recognition requirements of the core application scenes. And based on the data samples, the accuracy and operating speed of the address recognition methods are studied and analyzed, and the practicability of the algorithms is compared. Finally, the algorithms are summarized and the future algorithm improvement methods are prospected, which gives a reference for the practical application to the related businesses of power enterprises.

Key words: natural language processing; fuzzy matching; address recognition; power enterprises; big data

0　引　言

各项研究证据显示人类的活动信息基本与地理位置相关。在电力企业各项业务中同样如此,用电地址、报修工单、设备位置等等,地理位置均是其核心字段。目前常规模式基本都是人工提取各类信息中所含地址字段进行匹配,对于电力企业的相关工作人员均是较为繁琐、耗时的一项工作。如果能快速准确地识别相关内部业务和客户诉求中所含的地理位置信息,可以很好地辅助电力企业快速响应各方需求、提高服务能力、提升数据资产价值。

下面旨在探索实践如何自动快速准确地进行地理位置匹配,以各类常见信息中地理位置为基础,实现地理位置的自动提取与匹配,并基于Python编写多种地址提取匹配方法,综合考量匹配准确率及运行速度两类指标,探索较为适用于电力企业的方法。

1 研究背景

1.1 技术概述

最早的自然语言理解方面的研究工作是机器翻译。由美国人威弗于 1949 年首先提出了机器翻译设计方案。20 世纪 60 年代,国外对机器翻译曾有大规模的研究工作,耗费了巨额费用;但当时人们显然是低估了自然语言的复杂性,且语言处理的理论和技术均不成熟,机器翻译整体进展缓慢。当时的主要做法为存储两种语言的单词、短语对应制作大辞典,翻译时一一对应,技术上只是调整语言的同条顺序,但面对日常语义交流时,表现较差。

现在自然语言处理已经是计算机科学领域与人工智能领域中的一个重要方向。其研究的是如何实现人与计算机之间用自然语言进行有效通信的方法。自然语言处理本身为语言学、计算机科学、数学于一体。因此,自然语言处理的研究将涉及人们日常使用的语言。

例如,对于中文文本来说,其形式上由汉字(包括标点符号等)组成的一个字符串。字可组成词,词可组成词组,词组可组成句子,进而由一些句子组成段、节、章、篇。无论在上述各种层次的字(符)、词、词组、句子、段中,同样的词语在各层级之间都存在着歧义和多义现象,即形式上一样的一段字符串,在不同的场景或不同的语境下,可以理解成不同的词串、词组串等[1]。一般情况下,它们中的大多数都是可以根据相应的语境和场景的规定而得到解决的。也就是说,对于个人来说,并不造成无法理解的情况,这就是常见的联系上下文分析。但是为了正确理解语义,是需要基于极其大量的基础常识进行推理得出。而如何将这些基础常识较完整地进行收集和整理,又如何通过合适的方式,将其又快又好地存入计算机当中去,并且如何合理地利用它们来解析语义,都是工作量巨大且非常困难的工作。这不是一个在短时间内可以完成的工作,还有待长期的、系统的探索专研。

目前自然语言处理在应用领域包括“语言分词”“语义分析”“语言翻译”等等[2],而模糊识别是该领域重要的方法思路之一,其指在大量复杂的信息中,识别出有用的部份,即对接收的信息与以往的记忆和经验进行有关联认识,剔除无关的信息。无论是什么对象(语言、文字、图像等),其需要识别的核心点均具备自己特有的关键特征[3],对于中文地址来说,它的核心特征则是地址中的关键字,比如“市”“区”“街道”等等,常见方法则是基于关键字有效定位文章中地址位置,并基于分词、比对、校验等方式对地址模拟匹配,实现文章地址信息的提取[4]。

1.2 电力企业与地址相关业务现状

电力企业与地址相关的业务包括客户新装、报修以及设备安装管理等方面,除了目前地理位置信息存在不准确、不规范等问题外,还存在地址提取匹配耗时长、工作量大等问题。从场景应用上来看大致可以分为以下 3 类场景:

1)电力企业对内的检修、巡视、停电等工作情况录入。例如输电、变电等专业开展设备运行维护时,其当日工作情况需录入系统,特别是在发现故障时需在系统内录入故障内容(包括故障原因、故障地点等),该内容主要为人工填写。

2)电力企业对外的客户诉求处理。例如 95598 接单是根据客户诉求,形成工单明细内容,在工单派发前,需拟为工单标准格式,如图 1 所示,需要人工完善所属辖区、地址信息等内容。

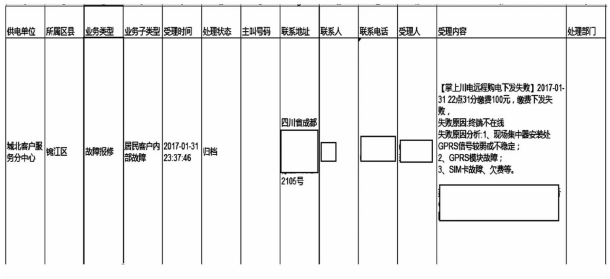

图 1 客户诉求工单样例

3)大数据应用,各专业管理部门根据专业管理需求,开展大数据分析,很多场景都需要结合地址信息开展相关工作。例如运检需要根据设备故障位置,制定下一步配电网可靠性提升方案;营销部需要根据投诉内容,针对性开展营销服务优质提升等。如果在没有准确地址信息的情况下,需要基于故障内容信息或者不规范的地址信息,通过人工观察定位的方式梳理地理位置,如果工作量较大,可能会一定程度上舍弃数据分析的颗粒度。

前两种应用场景是工作流程中必须涉及的工作内容,虽然看似简单,但是大量且固化的工作也在不经意地耗去了庞大的人力资源,而第 3 种情况,更是

会制约电力企业的精益化管理发展。

除此上诉 3 类场景之外,目前随着电力企业大数据进程的不断演进,针对地址信息进行的数据挖掘也在不断增多,因此现在亟需高效、准确的地址识别方法。

1.3 国内外研究现状

目前国内外在地址模糊匹配上,受制于语言的不同,国外项目课题难以直接应用。同时相关研究的主要内容是研究针对算法的准确性进行不断提升,而算法的复杂决定了其在运行上较为缓慢,并且有关算法较为针对特定场景及内容,不代表能直接在电力企业相关场景应用。下面将在各项主流方法的基础上,探索实践较为适用电力企业应用场景的地址匹配方法。

2 数据准备

2.1 文章信息准备

中文地址信息一般为“省 - 市 - 县 - 区 - 街道 - 街道号”,而目前日常经常接触的文章内容,主要包括以下几类样例:

1)网页信息填报类型。其表现为格式相对规范,基本不会出现多个地址信息,识别地址信息位置标志物明显,如图 2 所示。

基本信息

服务对象	自然人	办件类型	
法定办结时限	30 法定办结时限说明	法定办结时限单位	
承诺办结时限	5	承诺办结时限单位	
审批结果	审批结果类型	审批结果名称	
	证照	律师执业证	
是否收费	否	办理形式	
通办范围	无	特殊环节 (特别程序)	
办理地点	南安市柳城街道著名的柳小路6号		

图 2 网页信息填报样例

2)信息公开类型。一般为通知、公告,格式内容较为正式,地址信息较为清晰,但可能出现多个地址[5]。信息公开样例如图 3 所示。

3)文章信息类。一般为新闻报道、客户诉求等等,文字内容较多,地址信息不规范,存在多个地址信息等,该类情况相对较为复杂。例如“今日,在成都武侯区将进行计划检修停电,停电范围覆盖长寿路、桐梓林小区、成都中医院、四川省体育馆……成都晚报 12:00 发布消息”。

经过前期收集整理,收集文章地址信息 500 余

2020年疫情期开放时间调整详情

调整日期:从5月16日(本周六)开始

开放时间:调整为9:30-19:00

购票地点:广州风景秀丽的海珠中路33号

图源:本地宝 酷吧

部分游玩项目恢复开放:

怒族旋转屋、佤王府项目恢复开放,惊奇又刺激的怒族旋转屋等你来探险,在佤王府数码!

*具体演出时间以现场公告为准

温馨提示:

1、疫情期间,游客入园时须出示本人身份证等有效身份证件(与购票姓名相同);

2、进入景区时排队或游玩时,请与他人保持1.5米距离,服从现场工作人员指导;

3、若有发热、咳嗽等身体不适,请及时告知景区工作人员;

小编提示:微信搜索公众号【广州生活宝典

图 3 信息公开样例

条,整合了所述 3 种常见情况,使得匹配样本尽可能贴近真实应用场景,以确保算法在实际应用具有可操作性及实用性。

2.2 地址库信息准备

通过线上线下数据收集的方式,收集地址库 63 万条,在地址库样本范围上确保完全覆盖所需匹配的文章地址信息,并且加入相似、相近地址,进一步接近实际场景应用情况。地址库数据如图 4 所示。

province	city	district	township	street	street_num	locationx	locationy
安徽省	阜阳市	界首市	西城街道	界光路	561号	115.3678097	33.27176266
安徽省	阜阳市	颍州区	文峰街道	临泉路	428号	115.8003975	32.8978435
安徽省	阜阳市	颍州区	鼓楼街道	炮铺街	51号	115.811045	32.90747175
安徽省	阜阳市	颍州区	文峰街道	颍州中路	98号	115.8251926	32.8943316
安徽省	阜阳市	颍州区	文峰街道	清河东路	571-3号	115.8110317	32.89035383
安徽省	阜阳市	临泉县	城关镇	光明北路	199号	115.263916	33.06525
安徽省	阜阳市	阜南县	鹿城镇	谷河东路	42号	115.5918	32.638973
安徽省	阜阳市	颍泉区	中市街道	北京中路	103号	115.8297977	32.913523
安徽省	阜阳市	颍泉区	中市街道	慧湖路	115号	115.8202988	32.89625381
安徽省	阜阳市	阜南县	鹿城镇	谷河东路	113号	115.589262	32.636218
安徽省	阜阳市	颍泉区	中市街道	青年路	338号	115.833522	32.9029845
安徽省	阜阳市	颍州区	阜阳开发区京九街道	新安大道	297号	115.8487543	32.8615465
安徽省	阜阳市	颍州区	阜阳开发区京九街道	新安大道	927号	115.861156	32.84921
安徽省	阜阳市	颍州区	阜阳开发区京九街道	纬八路	829号	115.8492254	32.84022606
安徽省	阜阳市	临泉县	城关镇	解放路	45号	115.2541064	33.0612256
安徽省	阜阳市	界首市	东城街道	东升路	91号	115.3775075	33.26317216
安徽省	阜阳市	颍州区	鼓楼街道	一人巷商业街	4号	115.8123067	32.90026233
安徽省	阜阳市	颍州区	鼓楼街道	解放中路	35号	115.8139107	32.905427
安徽省	阜阳市	颍州区	文峰街道	中清路	12号	115.820033	32.888002
安徽省	阜阳市	太和县	城关镇	环城南路	2号楼	115.6239265	33.16294683

备注:本次算法测试运行配置为 2.6 GHz/CPU,内存 16GB。根据数据保密有关要求,地址库数据仅截图展示,同时此处展示的坐标已做调整,非真实坐标,但不会影响所研究内容。

图 4 地址库数据展示

3 算法思路及实现

3.1 方案 1:地址信息搜索加权

经验证直接利用正则表达式制定提取样式[6],难以实现地址的提取,特别是街道号,因此形成思路如下:将文章内容直接纳入地址库进行匹配,对应地址库每一行的 6 个字段,匹配上的字段为 1,没有匹配上的为 0,加总后除以 6 即为该地址的匹配率。最后找到匹配率最高的地址作为文章的匹配地址。基础代码实现如下:

数据读取:将地址库数据读入 Jupyter 运行环境,如图 5 所示。

```
#用时过长 需优化
import re
import pandas as pd

path = [redacted]
df = pd.read_excel(path)
df
```

	province	city	district	township	street	street_num	locationx	locationy	SUM1
0	安徽	阜阳	界首市	西城街道	界光路	561号	115.367810	33.271763	安徽省阜阳市界首市西城街道界光路
1	安徽	阜阳	颍州区	文峰街道	临泉路	428号	115.800398	32.897844	安徽省阜阳市颍州区文峰街道临泉路
2	安徽	阜阳	颍州区	鼓楼街道	炮铺街	51号	115.811045	32.907472	安徽省阜阳市颍州区鼓楼街道炮铺街
3	安徽	阜阳	颍州区	文峰街道	颍州中路	98号	115.825193	32.894332	安徽省阜阳市颍州区文峰街道颍州中路
4	安徽	阜阳	颍州区	文峰街道	清河东路	571-3号	115.811032	32.890354	安徽省阜阳市颍州区文峰街道清河东路
...	...	...	...	...	...	...	...	...	...

图 5　数据读取

逐一搜索加权:将 500 篇文章信息循环进行读取,并逐一将地址库字段纳入文章中进行比对,如果该字段在文章出现则该地址加 1,并选出数值最大的地址信息作为匹配结果,代码如图 6 所示。

```
for i in range(1,501):
    lt = []
    with open[redacted]
        txt1 = f.read()
```

```
ext = ["city","district","township","street","street_num"]
c2 = []
for i in range(1,501):
    lt = []
    with open[redacted]
        txt1 = f.read()
    c1 = []
    for i in range(0,635760):   #第一行
        c = 0
        for e in ext:
            if str(df[e][i]) in txt1:
                c = c + 1
        c1.append(c/6)
    c2.append(c1.index(max(c1)))
```

图 6　地址搜索加权实现

结果比对:循环抽取测试结果与正确答案表进行比对验证,计算正确结果数量以及算法准确率,代码实现及运行结果详见图 7。

```
import pandas as pd
path = r[redacted]
df_jg = pd.read_csv(path,encoding ="gbk")      #读取正确答案数据集
path = r[redacted]
df_cs = pd.read_csv(path)                      #读取运算结果数据集
n = 0
for x,y in zip(df_jg["label"].tolist(),df_cs["label"].tolist()):
    if x==y:                                   #答案一致，则n+1
        n = n + 1
print("测试结果中正确答案数量为",n,"准确率为",n/500)   #输出计算结果
```

测试结果中正确答案数量为 356 准确率为 0.712

图 7　方案 1 计算结果验证

从计算结果上来看,该方法的准确率为 71.21%,但从计算速度上,500 条文章完成所有匹配耗时为 4 h,平均每条耗时约 0.5 min,完全无法达到实用水平。

3.2　方案 2:模拟人工地址筛查

经过前一种方法的实验,准确率上虽然基本能够达到优化条件,但是运行效率低下,因此需要重新调整思路。经分析,方案 1 耗时长的主要原因为每条文章均需要与地址库 63 万条数据进行匹配,运算量大。实际可以无需每次进行 63 万条的匹配,一方面可以对字段进行去重,例如地址库的省只有 6 个、城市 80 个、县区 1000 个,因此可以通过不断定位省 - 市 - 县 - 区,逐步缩小匹配范围,以此大幅降低运算复杂程度。从上述思路来看,该方法基本类似于人工筛查的方式(逐个提取关键字段信息,然后在地址库中进行筛选,不断缩小范围实现最终定位)。其基本实现方式如下:

1)基于 Python 集合自带的去重功能,对地址库的省、市、县、区、街道、街道号等字段进行合并去重,形成地址库各字段的去重列表,代码如图 8 所示。

```
provice = []
for i in df["province"]:
    provice.append(re.findall("(.*)省 ",i)[0])
provice = list(set(provice))

city = []
for i in df["city"]:
    city.append(re.findall("(.*)市",i)[0])
city = list(set(city))

district = []
for i in df["district"]:
    district.append(i)
district = list(set(district))

street = []
for i in df["street"]:
    street.append(i)
street = list(set(street))

township = []
for i in df["township"]:
    township.append(i)
township = list(set(township))

street_num = []
for i in df["street_num"]:
    street_num.append(i)
street_num = list(set(street_num))

ext = ["city","district","township","street","street_num"]
```

图 8　核心地址字段去重

2)将地址库地址信息列表放入地址库中进行循环匹配,匹配成功则进行提取,形成匹配结果字典,代码如图 9 所示。

```
c1 = []
for c in city:     #判断地址是否在文章中出现
    if c in txt1:
        c1.append(c+"市")
d1 = []
for d in district:
    if d in txt1:
        d1.append(d)
tp1 = []
for tp in township:
    if str(tp) in txt1:
        tp1.append(str(tp))
s1 = []
for s in street:
    if str(s) in txt1:
        s1.append(str(s))
sn1 = []
for sn in street_num:
    if str(sn) in txt1:
        sn1.append(str(sn))    #匹配所有街道
snx = []
for i in sn1:
    snx.append(len(i))      #计算所有匹配到的数据
index =[]
for i in range(len(snx)):
    if max(snx) < 2:
        index.append(i)
    elif snx[i]>1:
        index.append(i)
sn11 = []
for i in index:
    sn11.append(sn1[i])
```

图 9　文章地址信息提取

3)按照匹配结果字典,根据匹配字段数量纳入地址库中进行定位,例如匹配到了“城市 - 街道 - 街道号”,则匹字段数字为 3,则在地址库对应 3 个字段进行 3 次筛查,逐步缩小匹配范围完成对地址位置的定位,详见图 10。

```
if len(jk)==2:
    a1 = []
    for i in jk_va[0]:
        a1.append(df[df[jk_keys[0]]== i])
    df1 = pd.concat(a1)
    if len(df1)>0:
        a2 = []
        for i in jk_va[1]:
            a2.append(df1[df1[jk_keys[1]]== i])
        df2 = pd.concat(a2)
        if len(df2)>0:
            c_2 = []
            for i in range(0,len(df2)):
                c1.update({df2.index.tolist()[i]:len(df2["SUM1"].tolist()[i])})
            max_value = max(c1.values())
            for m, n in c1.items():
                c_2.append(m)
            if len(c_2)>1:
                cf = []
                for i in range(0,len(c_2)):
                    df["SUM1"][c_2[i]]
                    jb = jieba.lcut(df["SUM1"][c_2[i]])
                    cf.append(len(jb) - len(set(jb)))
                if len(set(cf)) ==1:
                    bqd_dict.update({page:c_2})
                c2.append(c_2[cf.index(min(cf))])
            else:
                c2.append(c_2[0])
        else:
            c1.update(({len(df1["SUM1"].tolist()[0]):df1.index.tolist()[0]}))
            c2.append(c1[max(c1.keys())])

if len(jk)==3:
    a1 = []
    for i in jk_va[0]:
        a1.append(df[df[jk_keys[0]]== i])
    df1 = pd.concat(a1)
    if len(df1)>0:
        a2 = []
        for i in jk_va[1]:
            a3.append(df2[df2[jk_keys[2]]== i])
        df3 = pd.concat(a3)
        if len(df3)>0:
            c_2 = []
            for i in range(0,len(df3)):
                c1.update({df3.index.tolist()[i]:len(df3["SUM1"].tolist()[i])})
            max_value = max(c1.values())
            for m, n in c1.items():
                c_2.append(m)
            if len(c_2)>1:
                cf = []
                for i in range(0,len(c_2)):
                    df["SUM1"][c_2[i]]
                    jb = jieba.lcut(df["SUM1"][c_2[i]])
                    cf.append(len(jb) - len(set(jb)))
                if len(set(cf)) ==1:
                    bqd_dict.update({page:c_2})
                c2.append(c_2[cf.index(min(cf))])
            else:
                c2.append(c_2[0])
        else:
            c1.update(({len(df2["SUM1"].tolist()[0]):df2.index.tolist()[0]}))
            c2.append(c1[max(c1.keys())])
```

图 10　根据地址提取信息进行筛选匹配

4)验证该方法的准确性,详见图 11。

```
import pandas as pd
path = r"...
df_jg = pd.read_csv(path,encoding ="gbk")      #读取正确答案数据集
path = r"...
df_cs = pd.read_csv(path)                      #读取运算结果数据集
n = 0
for x,y in zip(df_jg["label"].tolist(),df_cs["label"].tolist()):
    if x==y:                                   #答案一致，则n+1
        n = n + 1
print("测试结果中正确答案数量为",n,"准确率为",n/500)   #输出计算结果
```

```
测试结果中正确答案数量为 325 准确率为 0.65
```

图 11　方案 2 计算结果验证

该方案在运行速度上提升明显,500 篇文章基于 63 万条地址库的匹配,运行完成仅需 5 min,平均每篇文章匹配仅需 0.6 s,在运行速度上完全满足实用化要求,但是在准确率上,现阶段该方案也仅有 65% 左右,无法达到实用化水平(一般设计算法,置信率要达到 95% 以上)。

解决运行速度慢的问题后,总结错误匹配原因,逐个解决错误匹配问题,进行算法优化。

经逐个分析,出现错误的原因有以下几点:

1)地址信息相似性极高的情况下,出错率高。例如 xx 区 - 城西街道 16 号与 xx 区 - 城西街道 6 号。

2)可能会筛查出两个以上答案。例如文章为城西 5 号,地址库为城西区城西街道、城西区城西 5 号。

3)个别答案匹配不上。例如文章为成都武侯区 xx 街道(同时西安市也有该街道名的情况),可能就会因匹配顺序问题导致结果无法正确输出(因为城市与街道无法对应的问题)等,错误匹配结果如图 12 所示。

filenam	label	原文
174.html	福建省$福州市$长乐区$吴航街道$胜利路$114号$1胜利路114号	
180.html	福建省$泉州市$丰泽区$东湖街道$东湖街$298号$1丰泽区东湖街道风景秀丽的湖心街298号	
223.html	安徽省$合肥市$蜀山区$芙蓉社区$翡翠路$翡翠路$翡翠路385号	
320.html	广东省$广州市$白云区$均禾街道$新石路$新石路$白云区新石路79号	

图 12　错误结果样例

3.3　方案 3:模拟人工地址筛查(场景问题优化)

对于地址雷同、重复等问题导致匹配不准确的情况,常规解决方式可以在匹配前进行分词;但是由于地址名称复杂度高、非常用名称多[7],常用的 cpca 库无法使用,即使是分词效果最为良好的 jieba 分词方法[8],依然难以对地址名称进行准确分词。因此只能进行场景设计解决。

由于所有匹配不准确的地址问题均出在“街道号”这一字段,可以从“街道号”的字符长度进行处理,伪代码实现:

1)如果最长的街道号也不足 1,则直接纳入需要匹配的结果;

2)如果不是均为短值,则将街道号长度在 2 以上的数据纳入需要匹配的结果,避免过多无效数据进入需要匹配的内容;

3)做进一步判断,避免如 3 号、33 号的重复出现。

代码实现如图 13 所示。

对于存在多个地址信息的情况下,例如成都 - 64 号、西安 - 223 号,前后不对应的问题,解决思路为:定位匹配到的街道在文章中出现的位置以及街道号所出现的位置,比较两者位置距离,两者相近的为一组,就可以实现地址前后字段的准确关联。伪

```
sn1 = []
for sn in street_num:
    if str(sn) in txt1:
        sn1.append(str(sn))    #匹配所有街道号
snx = []
for i in sn1:
    snx.append(len(i))      #将街道号长度纳入snx，用作下一步判断
index =[]
for i in range(len(snx)):
    if max(snx) < 2:      #如果最长的街道号也不足1，则直接纳入需
        index.append(i)
    elif snx[i]>1:       #如果不是均为短值，则将街道号长度在2以上
        index.append(i)
sn11 = []
for i in index:          #做进一步判断，避免3号、33号的重复出现
    sn11.append(sn1[i])

n = len(sn11)
lt2 = []
for i in sn11:
    lt3 = []
    for c in range(0,n):
        if i in sn11[c]:
            lt3.append(1)
        else:
            lt3.append(0)
    lt2.append(sum(lt3))
n = -1
sn111 = []
for i in lt2:
    n = n + 1
    if i <2:
        sn111.append(sn11[n])
```

图 13　文章地址提取预处理

代码如下：

1）找到所需匹配的街道在文章中的位置 strlc；

2）找到街道对应位置最近的街道号码；

3）提取街道 - 街道号码进行组合；

4）将组合好的结果纳入地址库进行筛查。

匹配文章中出现的街道名称，并定位街道名称在文章出现的位置，基于该位置找到离街道名称最近的街道号码，代码实现如图 14 所示。

```
strlc = []
if len(c1)>1:   #找到所需匹配的街道在文章中的位置strlc
    for i in c1:
        strlc.append(txt1.find(i))
else:
    strlc.append(txt1.find(c1[0]))
jk = {}
num = []
if len(c1)>1:
    n = 0
    for i in strlc:
        num.append(re.findall('(\d+)',txt1[i:i+8]))              #25个字符修改为8字符
        jk.update({c1[n]:re.findall('(\d+)',txt1[i:i+8])})
        n = n + 1
else:
    num.append(re.findall('(\d+)',txt1[strlc[0]:strlc[0]+8]))
    jk.update({c1[0]:re.findall('(\d+)',txt1[strlc[0]:strlc[0]+8])})

for k in list(jk.keys()):    #去除没有街道号的街道名称
    if not jk[k]:
        del jk[k]
```

图 14　定位街道字符串在文章中的位置并找到最近街道号

地址提取的运行结果如图 15 所示，而未优化前的运行结果如图 16 所示。

通过对比，可以看到算法优化后，大幅减少了匹配到的街道号码，同时针对有多个地址的文章实现了街道名称与街道号码的一一对应，可以避免出现上下不对应导致无法匹配的情况。

计算结果验证如图 17 所示。经过验证，虽然在

```
{'溪源宫路': ['96']}
{'开禾路': ['88']}
{'长虹南路': ['31']}
{'胜利路': ['117', '7']}
{'八七路': ['1056']}
{'大东街': ['49']}
{'南洋路': ['1819']}
{'安福西路': ['307']}
{'天波路': ['84']}
{'北环东路': ['8', '10']}
{'前进路': ['53']}
{'白泉街': ['19']}
{'下杭路': ['35', '37']}
{'城东路': ['110']}
{'云顶中路': ['98'], '金山街': ['3']}
{'盘龙大道': ['141']}
{'滨江大道': ['154']}
{'金尚路': ['202', '2']}
```

图 15　优化后实现“街道 - 街道号码”相对应

```
36 {'township': ['冶父山镇'], 'street_num': ['11', '17', '46', '45', '10', '47',
57 {'street': ['国货西路'], 'street_num': ['11', '46', '14', '201', '6号', '46号
58 {'street': ['岗东路', '龙岗东路'], 'street_num': ['15', '27', '27号', '7号']}
61 {'street': ['赤岗北路'], 'street_num': ['15', '18', '8号', '18号', '赤岗']}
69 {'street': ['太南路', '沙太南路'], 'street_num': ['30', '10', '0号', '330号',
72 {'street': ['西洋中路', '洋中路'], 'street_num': ['18', '10', '3号', '183号',
92 {'street': ['石潭路'], 'street_num': ['16', '19', '30', '202', '8号', '38号']
94 {'street': ['蕙兰路'], 'street_num': ['16', '17', '19', '18', '23', '29', '30
96 {'street': ['直街', '龙新直街'], 'street_num': ['16', '19', '30', '202', '4号
99 {'street': ['镇龙里'], 'street_num': ['16', '19', '30', '202', '镇龙', '7号']
109 {'street': ['仙岳路'], 'street_num': ['11', '17', '23', '46', '45', '10', '6
132 {'street': ['洪山园路'], 'street_num': ['16', '24', '29', '30', '45', '10',
133 {'street': ['固镇路'], 'street_num': ['11', '17', '46', '45', '10', '47', '8
135 {'street': ['公园路'], 'street_num': ['10', '14', '48号', '48号楼', '148号',
136 {'street': ['内蒙路'], 'street_num': ['10', '1号']}
```

图 16　优化前多个街道和街道号结果不对应且未精确定位

运行速度上，优化后的算法运行需要 10 min，是未优化前的 2 倍，但是也基本满足应用需求，同时优化后准确率提升到了 98.6%，完全满足实际应用需要。

```
import pandas as pd
path = r"[redacted]
df_jg = pd.read_csv(path,encoding ="gbk")     #读取正确答案数据集
path = r"[redacted]
df_cs = pd.read_csv(path)                     #读取运算结果数据集
n = 0
for x,y in zip(df_jg["label"].tolist(),df_cs["label"].tolist()):
    if x==y:                                  #答案一致，则n+1
        n = n + 1
print("测试结果中正确答案数量为",n,"准确率为",n/500)  #输出计算结果

测试结果中正确答案数量为 493 准确率为 0.986
```

图 17　方案 3 计算结果验证

除以上 3 种方案外，前期还通过查阅相关文献，并结合课题内容进行实践，淘汰了纯正则表达式法提取、cpca 地址库提取等方法。第 3 种方案在准确率和运行速度上基本满相关业务场景应用需求。同时基于 Python 的 Pyinstaller 代码封装，可实现在电力企业各类工作电脑上直接运用，具备较高的可操作性及灵活性。3 种方案综合对比如表 1 所示。

表 1　3 种方法的综合对比

算法	准确率/%	运行速度	综合评价
地址搜索加权	71	4 h	速度慢、准确度低
模拟人工筛查	65	5 min	速度极快、准确率低
模拟人工优化	98.6	10 min	速度快、准确率高

（下转第 85 页）

[10] 王鸿玺,唐如意,吴一敌,等. 基于 HPLC 的智能抄表技术在客户侧泛在电力物联网中的研究及应用[J],电力系统保护与控制,2020,48(3):92 – 97.
[11] 陶鹏, 王鹏, 张艳. 不同通信方式对用电信息采集系统采集成功率的影响分析[J]. 河北电力技术, 2017(1): 3 – 4.
[12] 王凤祥,梁波,孔晶,等. 宽带载波技术应用及效益分析[J]. 国网技术学院学报, 2018, 21(3): 35 – 38.
[13] 周春良. 电力专用宽带电力线载波通信芯片的设计与应用[J]. 信息技术与网络安全, 2017, 36(11): 34 – 36.
[14] 薛晨. 中压配电网电力线载波通信系统自适应频点选择算法[J]. 电力系统保护与控制, 2018, 46(6): 24 – 29.
[15] 徐文涛. 基于 HPLC 通信技术的用电信息采集系统设计[J]. 微型电脑应用,2020, 35(12): 117 – 120.
[16] 徐文涛. 基于 HPLC 通信模块的智能电表改进分析[J]. 微型电脑应用,2019,35(11): 156 – 158.
[17] 李睿. 低压配电网电压质量问题分析与治理[J]. 电工技术,2020(1):110 – 112.

作者简介:

余 敏(1987),女,工程师,硕士,主要从事电能计量技术和用电信息采集技术的应用研究。

(收稿日期:2020 – 10 – 23)

(上接第 48 页)

4 结 语

经过测试,所提的地址提取匹配方法具备一定的实用性,可以应用于电力企业相关业务场景,但同时该技术主要面向于地址库中包含的地理位置信息,在此条件下能具备一定的准确性、运行速度和灵活优势。

在缺点方面,该方案对于日常可能会遇到的地址信息未包含在地址库内的情况没有进行设计处理,因此未来可以在上述方案的基础上进一步加入“最近地址提取法”[9]。现阶段形成思路如下:1)基于独热编码方式对地址库各地址信息进行解析,将文字进行量化处理;2)将处理后的地址库按区县进行分区,减少模型运算量,增加模型数量;3)基于随机森林、人工神经网络等方式定位分区中地址库每个字段的经纬度权重[10];4)对未能直接匹配成功的文章地址信息,进行独热化处理,并按照所属区域纳入模型进行计算,得出其经纬度;5)根据模型计算的经纬度,计算与地址库之间欧式距离或者曼哈顿距离找到最相近地址[11]。下一步将继续提升所提方案的全面性及可用性,为电力企业在管理及服务等方面应用提供参考。

参考文献

[1] 陈开昌. 自然语言处理技术中的中文分词研究[J]. 信息与电脑(理论版),2016(19):61 – 63.
[2] 韩程程,李磊,刘婷婷,等. 语义文本相似度计算方法[J]. 华东师范大学学报(自然科学版),2020(5):95 – 112.
[3] 张磊. 文本分类及分类算法研究综述[J]. 电脑知识与技术,2016(34):225 – 226.
[4] 王春柳,杨永辉,邓霏,等. 文本相似度计算方法研究综述[J]. 情报科学,2019(3):158 – 168.
[5] 徐培治,刘晓春,秦首科,等. 网页信息提取方法和装置:CN108399167A[P]. 2018 – 08 – 14.
[6] 王瑞波,王钰,李济洪. 面向文本数据的正则化交叉验证方法[J]. 中文信息学报,2019(5):54 – 65.
[7] 张大威. 微博数据的地理位置信息提取方法:CN105069071A[P]. 2015 – 11 – 18.
[8] 石凤贵. 基于 jieba 中文分词的中文文本语料预处理模块实现[J]. 电脑知识与技术,2020,16(14):254 – 257.
[9] 刘德喜,聂建云,张晶,等. 中文微博情感词提取:N – Gram 为特征的分类方法[J]. 研究中文信息学报,2016(4):193 – 205.
[10] 徐聪,张丰,杜震洪,等. 基于哈希和双数组 trie 树的多层次地址匹配算法[J]. 浙江大学学报(理学版),2014(2):217 – 222.
[11] 李想. 基于社交媒体的灾害事件提取与时空分析——以地震灾害为例[D]. 兰州:兰州交通大学,2018.

作者简介:

刘浩宇(1991),男,硕士,工程师,从事数据质量管理、电力大数据分析等工作;

李 喆(1989),男,硕士,工程师,从事 5G、电力北斗技术试点应用等工作;

佘佐超(1990),男,助理工程师,从事信息系统安全防护工作。

(收稿日期:2020 – 10 – 09)